Mixed safety critical system design and analysis

Invited Speaker: Rolf Ernst
TU Braunschweig, Germany

Overview

- motivation
- safety critical embedded system design
- networks and multi-core systems for mixed time and safety critical applications
- reliable systems for higher safety requirements
- example projects
- conclusion

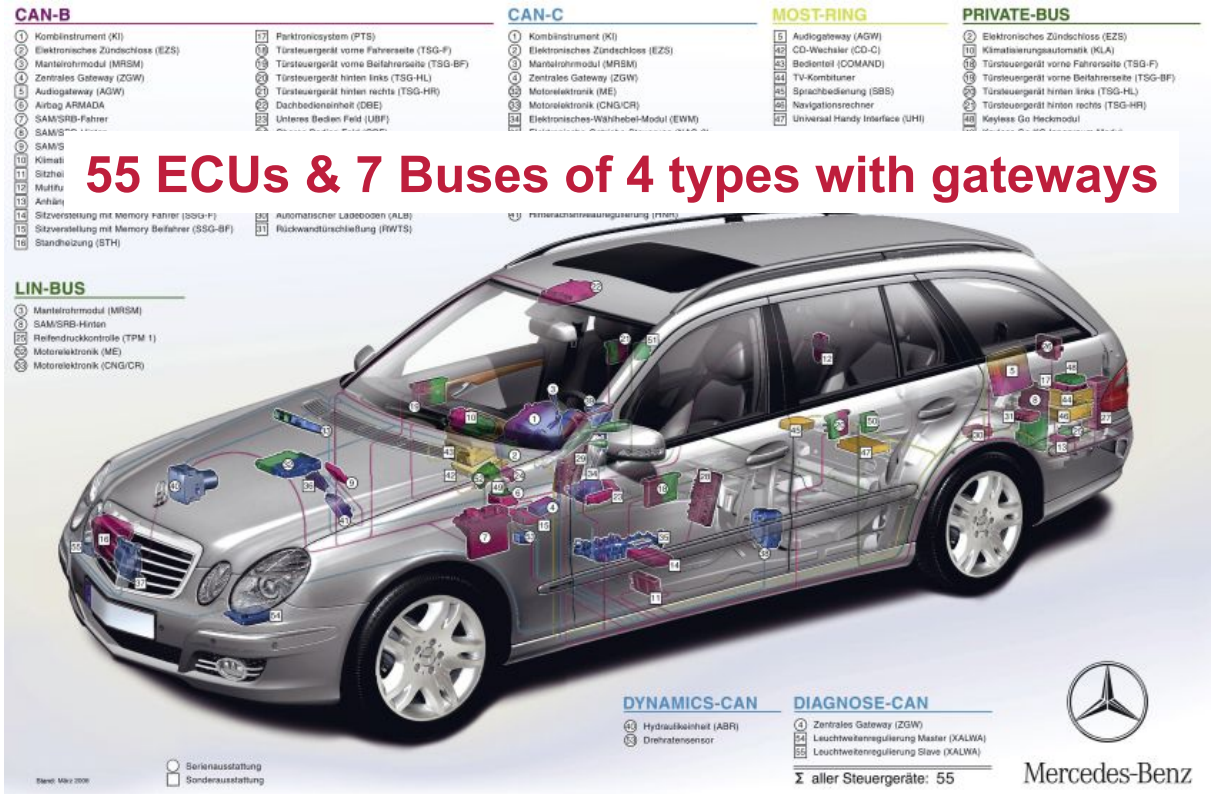
Overview

- **motivation**
- safety critical embedded system design
- networks and multi-core systems for mixed time and safety critical applications
- reliable systems for higher safety requirements
- example projects
- conclusion

Motivation

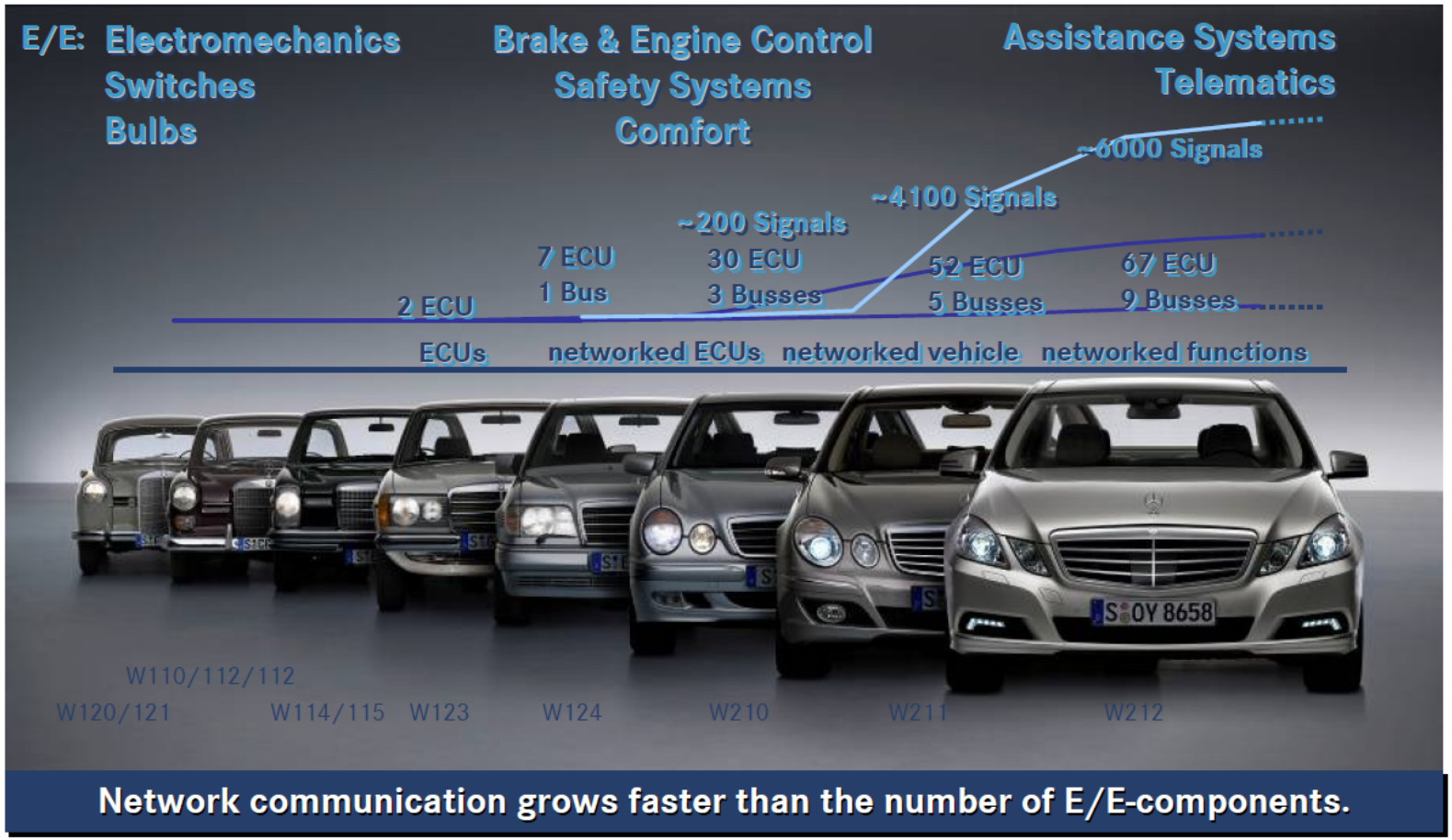
- today's embedded systems use complex networks
  - hundreds of functions
  - thousands of tasks
  - 50+ ECUs
  - networked control
  - many suppliers
  - heterogeneous

- networks are an efficient platform for systems integration



source: Daimler

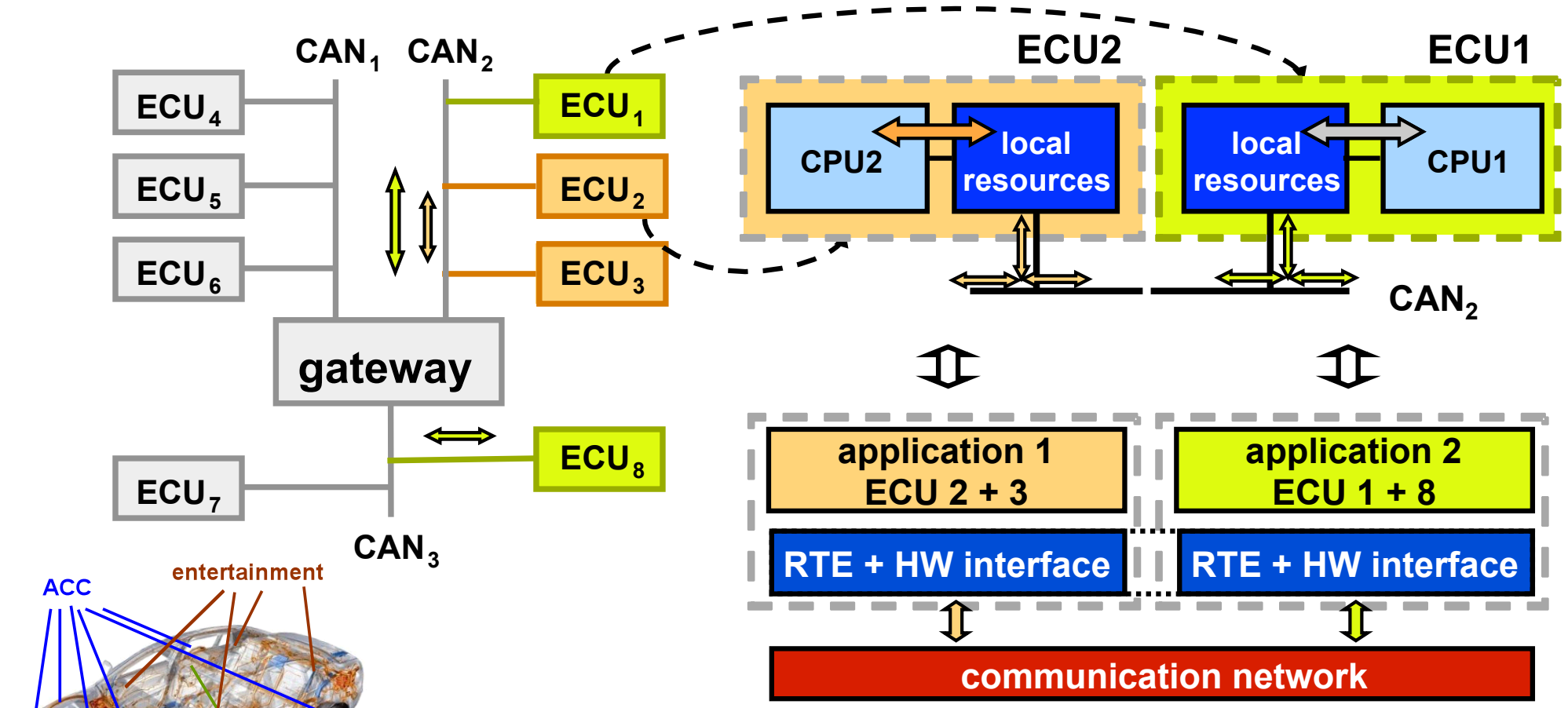
Growing Network Complexity



source: T. Bone, Daimler

Motivation

- networks lead to component sharing and networks for different functions



Automotive design chain – Many players

OEM

- BMW, Daimler, GM, PSA, Toyota, …
- global system, integration and network

specs ↓ ↑ ECUs

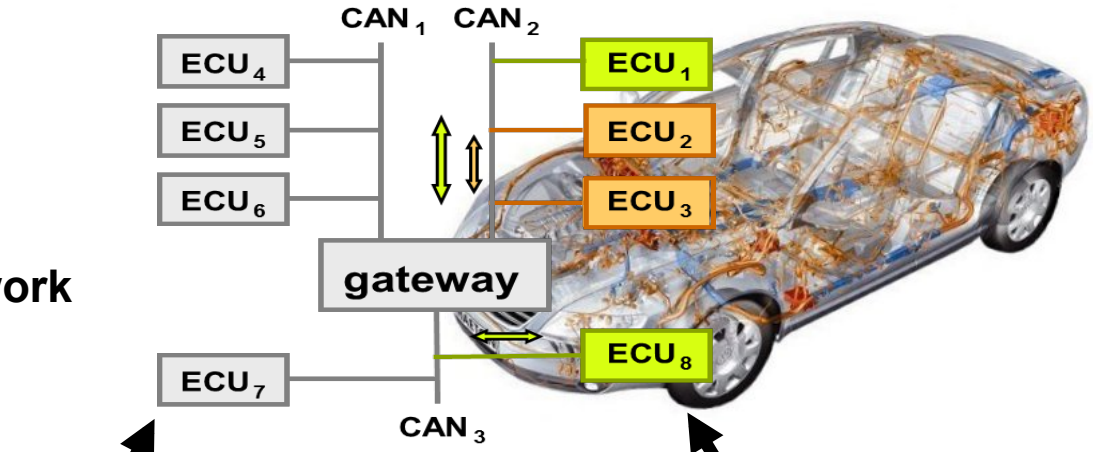
ECU - Supplier

- Bosch, Delphi, Valeo, …
- ECU responsibility

specs ↓ ↑ SoCs

HW Component - Supplier

- Infineon, Freescale, ST, Toshiba, …



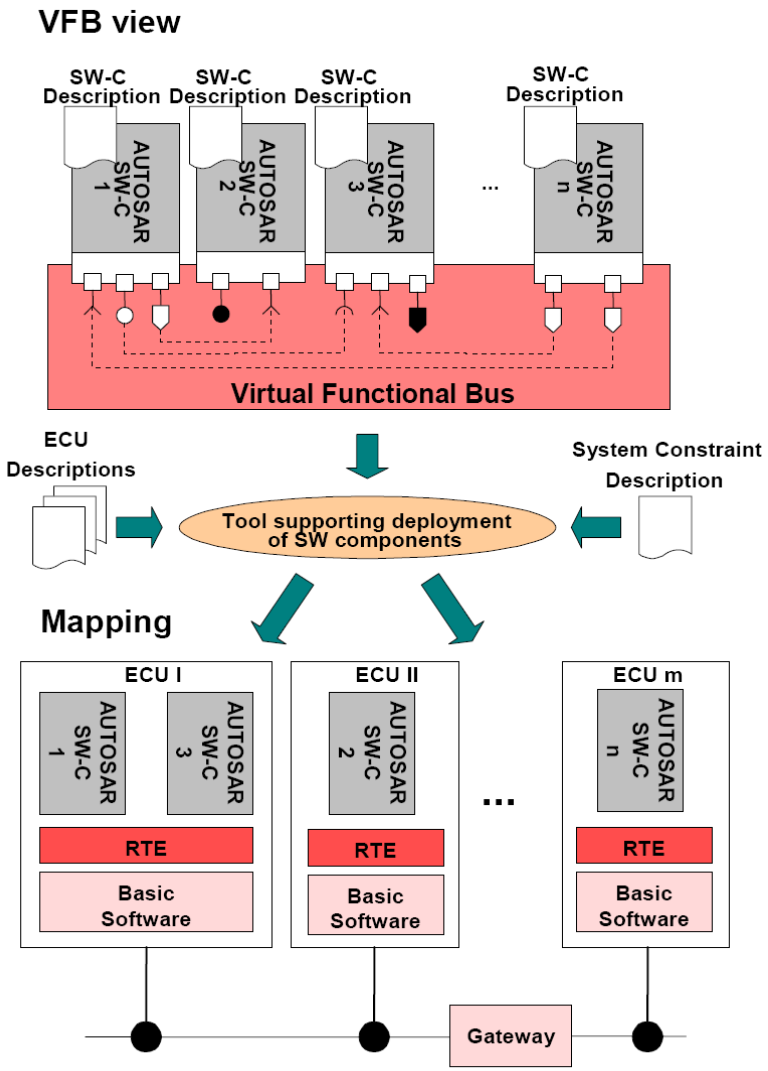
Bosch Delphi Valeo

RTE - Supplier

- Vektor, ETAS, Elektrobit, Mentor, …

Software standardization

- objectives
  - reuse and portability of applications
  - system optimization
  - defined interfaces for supply chain with standardized methods and tools
- example AUTOSAR
  - automotive standard software architecture
  - virtual functional bus for integration
  - run time environment (RTE) for specific ECUs



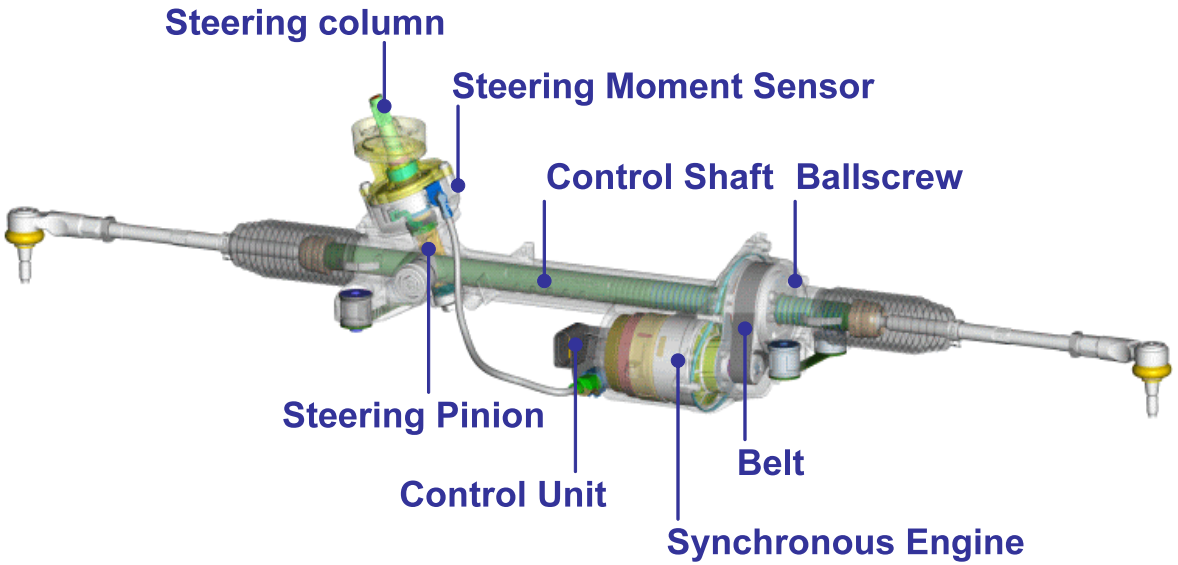
The safety challenge

- embedded systems are increasingly used to
  - implement advanced system features
  - improve safety
- in such cases, the embedded system inherits the safety and dependability requirements of the system function
  - safety related embedded systems
- such functions are no longer simple
- example: automotive electronics
  - electronic steering
  - camera based object recognition and tracking

Example 1: Electronic steering

- standard equipment functions
  - steering power support, speed dependent
  - active centering and dampening
  - straight-running function …
- upgrade equipment functions
  - park assist
  - lane-keeping assist
  - customizable adaptivity - from sportive to an emphasis on comfort

source: Volkswagen



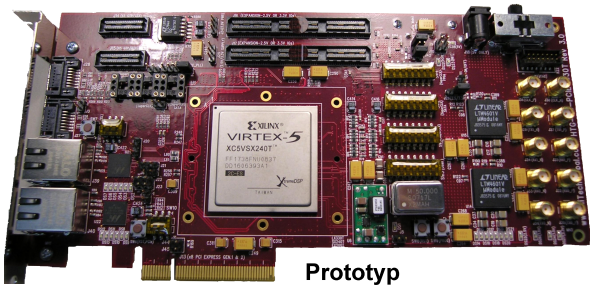
Embedded systems architecture

- two-computer system of the steering control unit
  - steering functions, motor control, and I/O handling are implemented on the main computer
  - the second computer monitors the main computer
  - communication via digital interface
  - exchange of high-frequency question-answer-sequences
  - both computers have an independent clock and energy supply

- classification: fail-safe system function – SIL 3 (more later)

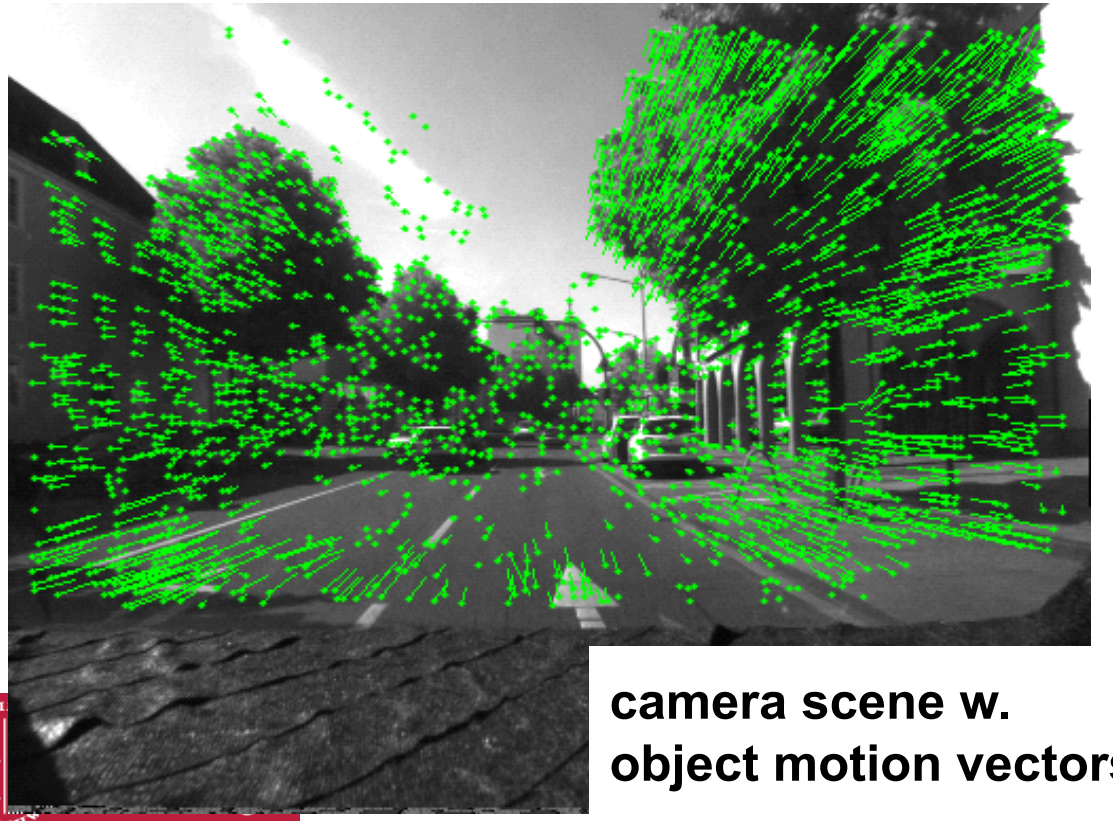
Example 2: Object recognition and tracking

- may be used as safety feature (collision avoidance) – SIL3?
- FPGA (or multi-core DSP)
- more than 100 GOp/s (algorithmic)
- power constrained (temperature)

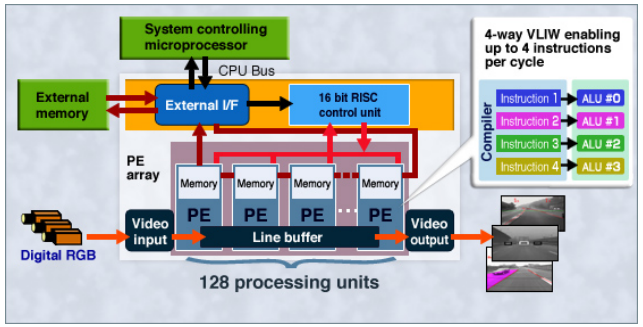


Prototyp
Hardware-Beschleuniger

FPGA prototype
source: NFF

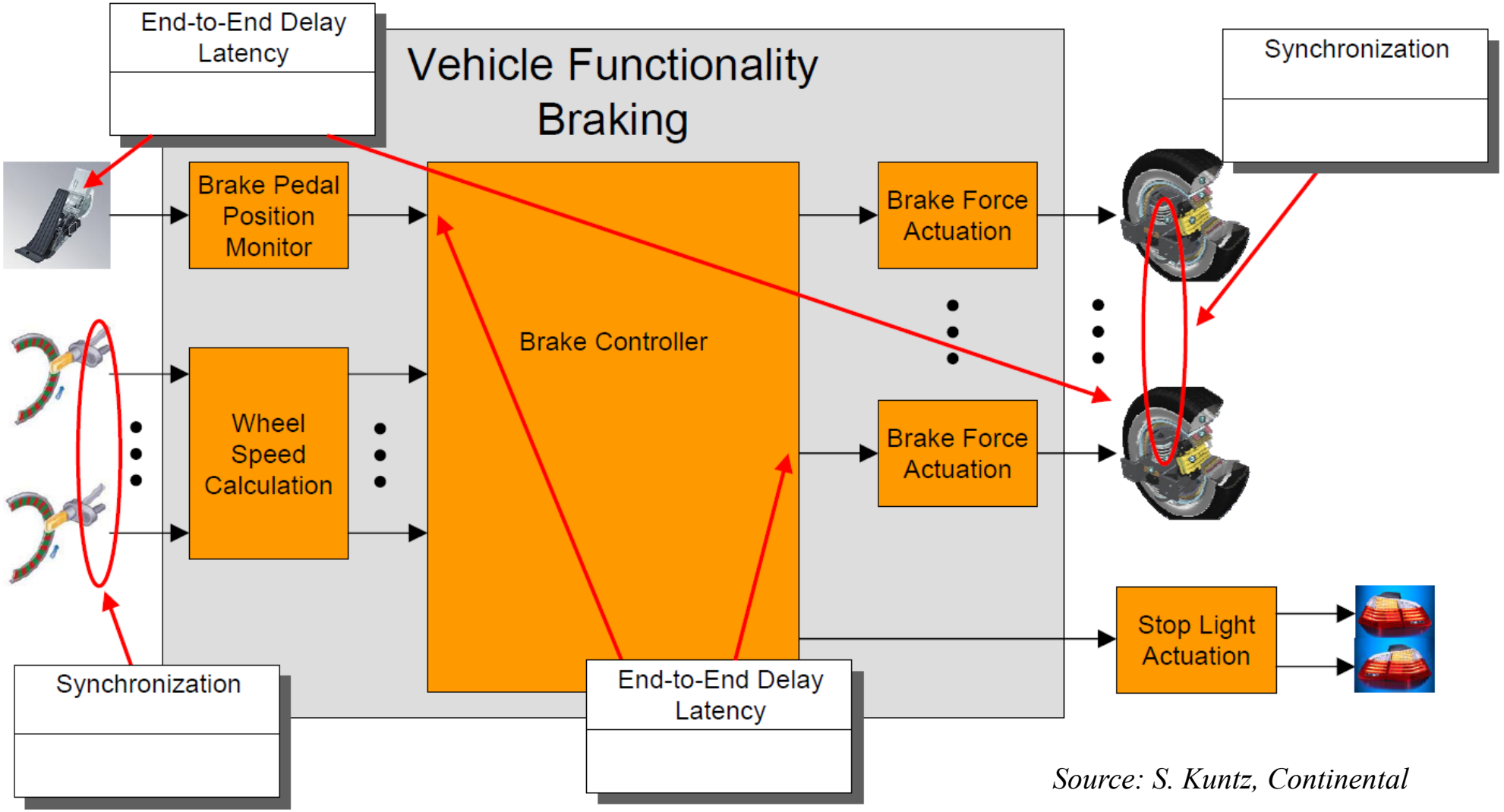


camera scene w.
object motion vectors



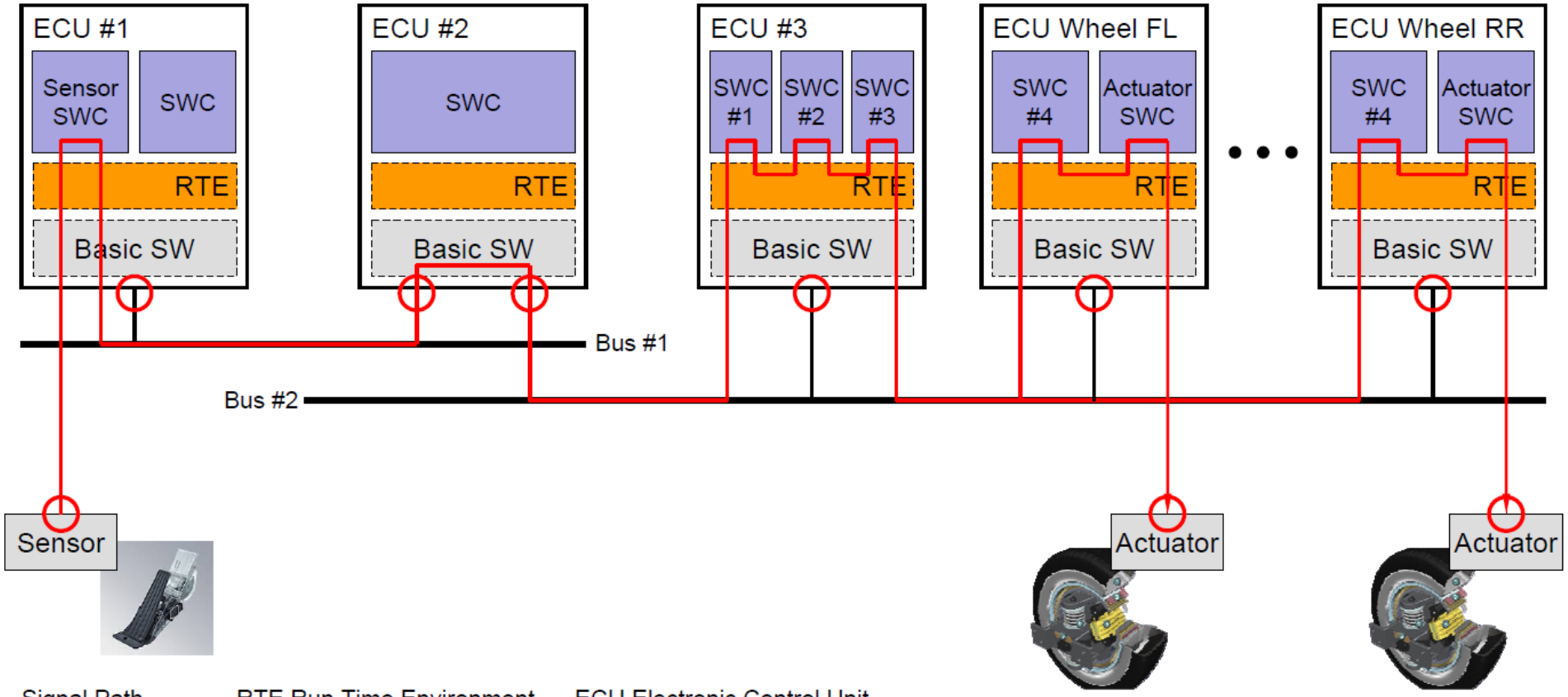
IMAPCAR DSP
(source: Renesas)

Safety functions are distributed



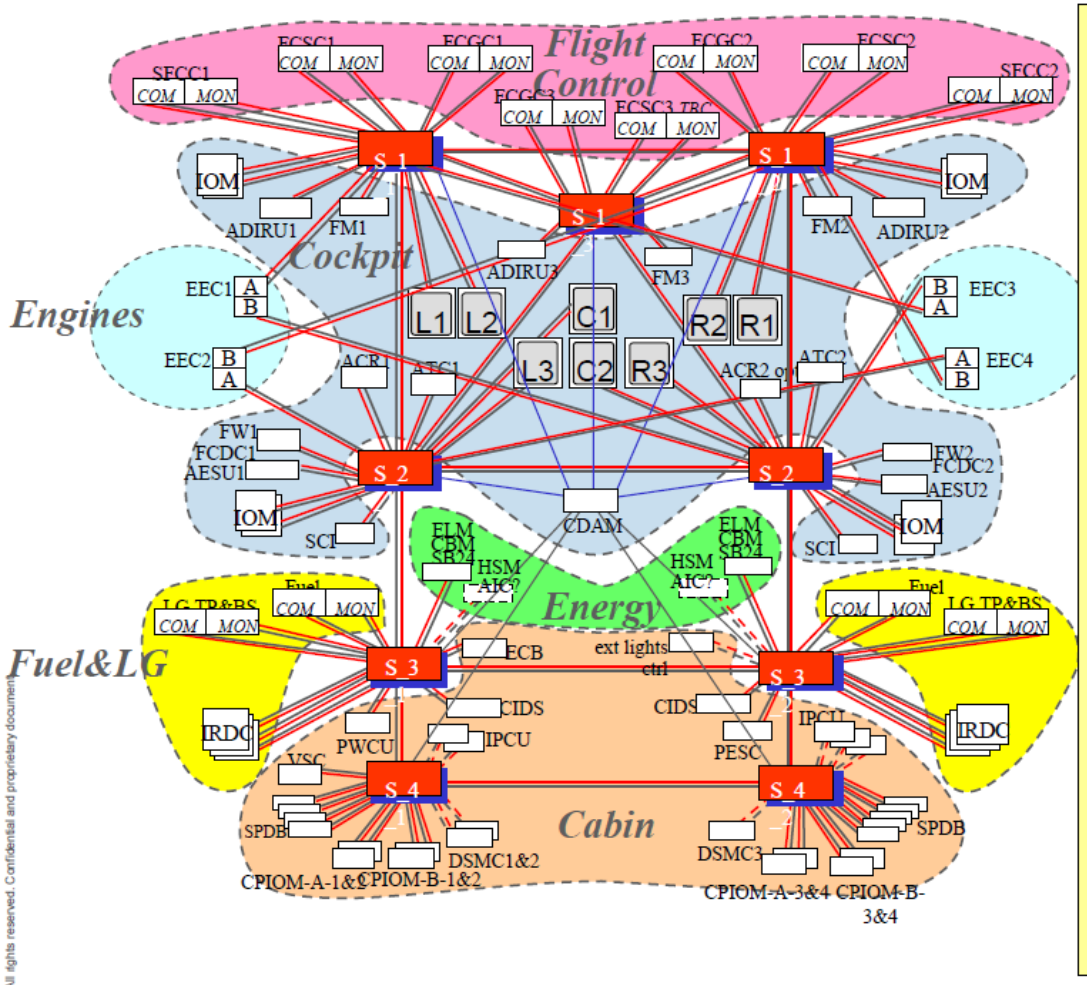
Source: S. Kuntz, Continental

Distributed brake function



Source: S. Kuntz, Continental

Avionics: Airbus 380 – AFDX based network

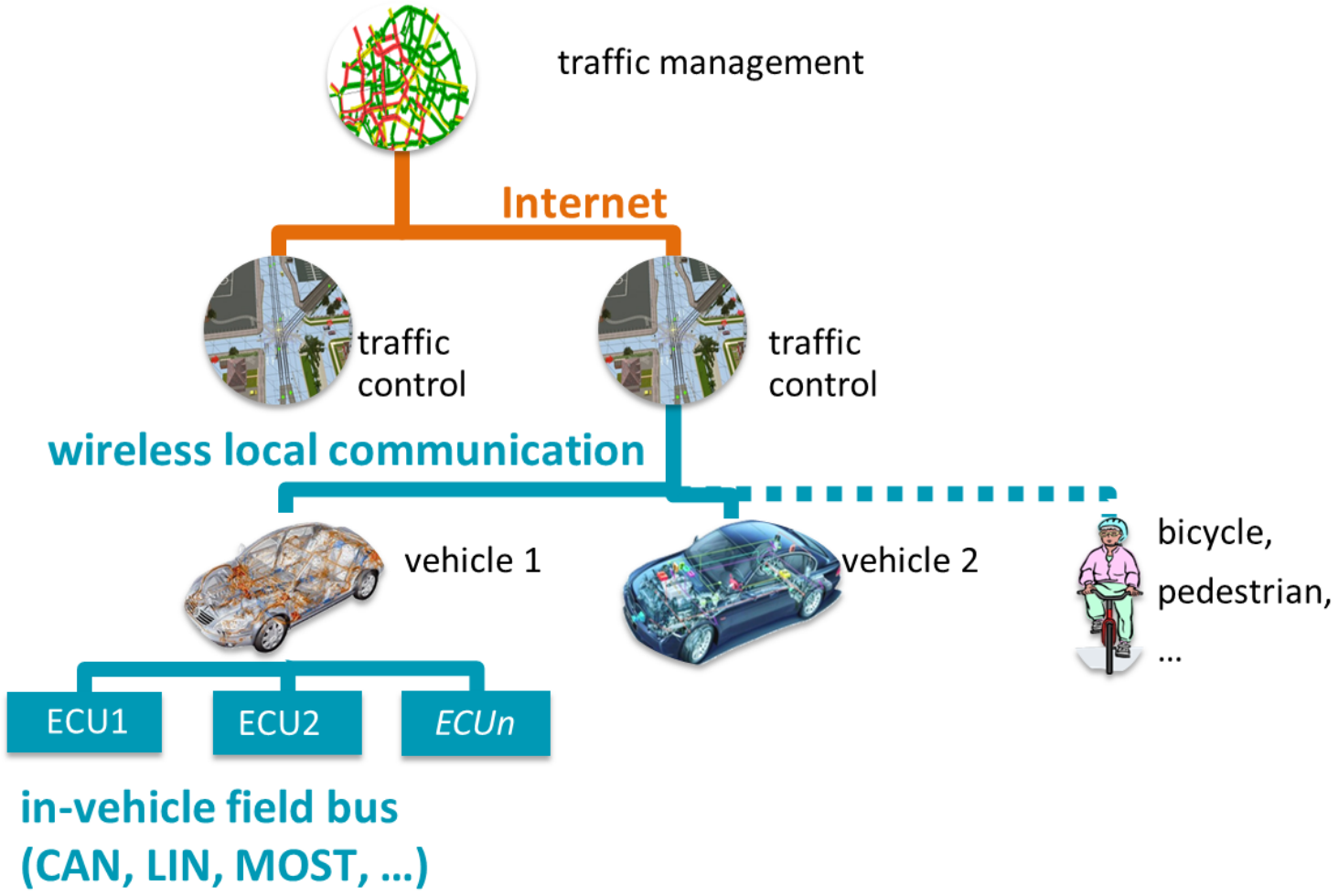


AFDX Network:

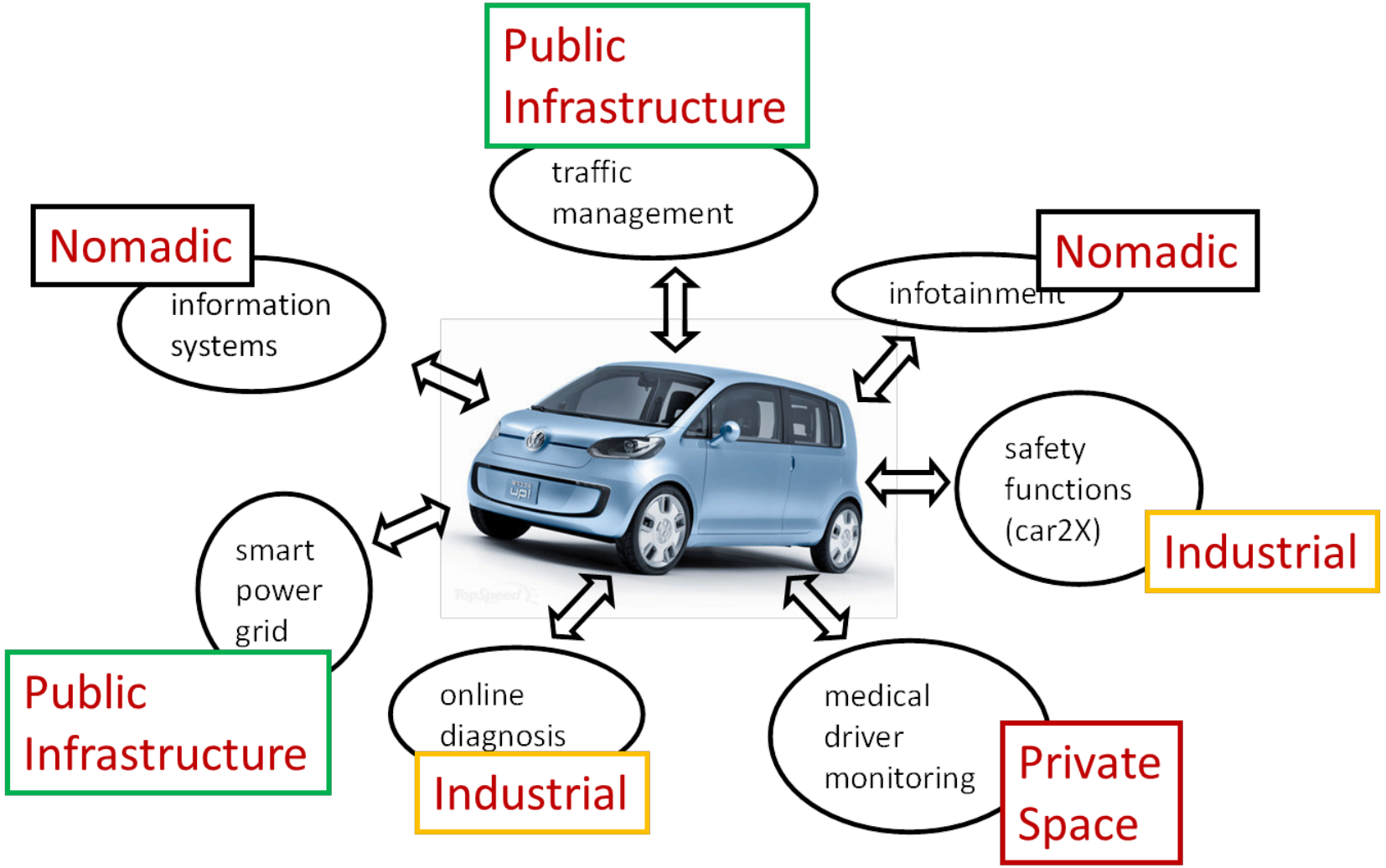
- 100 Mbits
- Redundant Network (A&B) with independent alimentation
- AFDX switches = 2 x 8
- NB of ports (connections) possible on each switch (20-24)
- MTBF of the switch is very high (100 000 hours expected)
- Up 80 AFDX subscriber

Source: J-B. Itier. A380 Integrated Modular Avionics Artist2 meeting 2007

Safety critical applications extended to open networks – Example traffic (ARTEMIS SRA)



Integration covers several industrial sectors



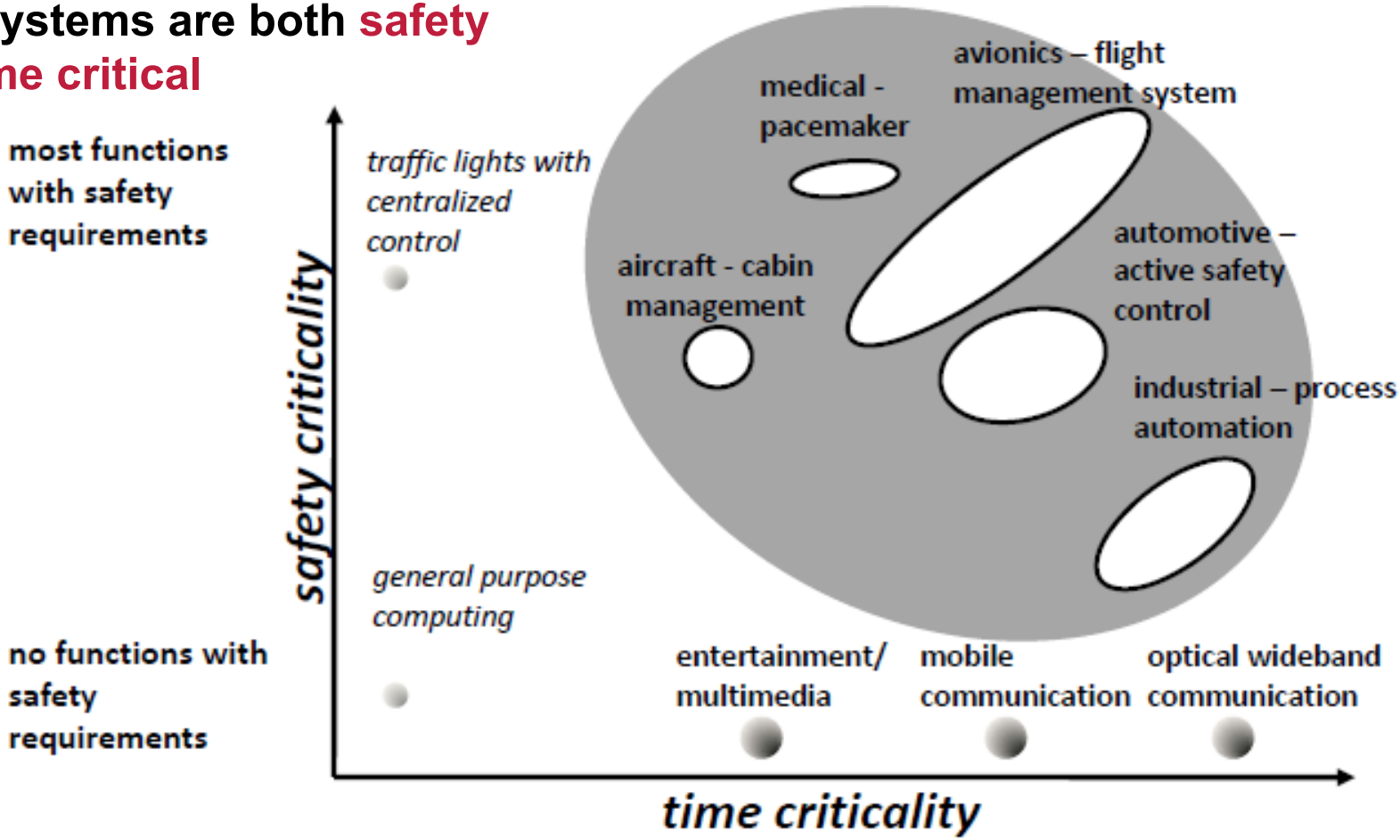
Merging functions with different criticality levels

- integration on one platform leads to systems with applications of different safety requirements
  - strict separation too expensive
  - mixed (safety) criticality systems
- mutual dependency via platform and sensors/actuators requires safety concept and qualification/certification for all functions
  - data often missing for less safety critical functions
  - high cost for qualification process of all applications on a platform
  - significant limitation and costs for updates

→ safety is highly relevant aspect in embedded systems integration

Safety and time criticality

- many safety critical systems have hard deadlines
  - such systems are both safety and time critical



Overview

- motivation
- **safety critical embedded system design**
- networks and multi-core systems for mixed time and safety critical applications
- reliable systems for higher safety requirements
- example projects
- conclusion

Safety standards

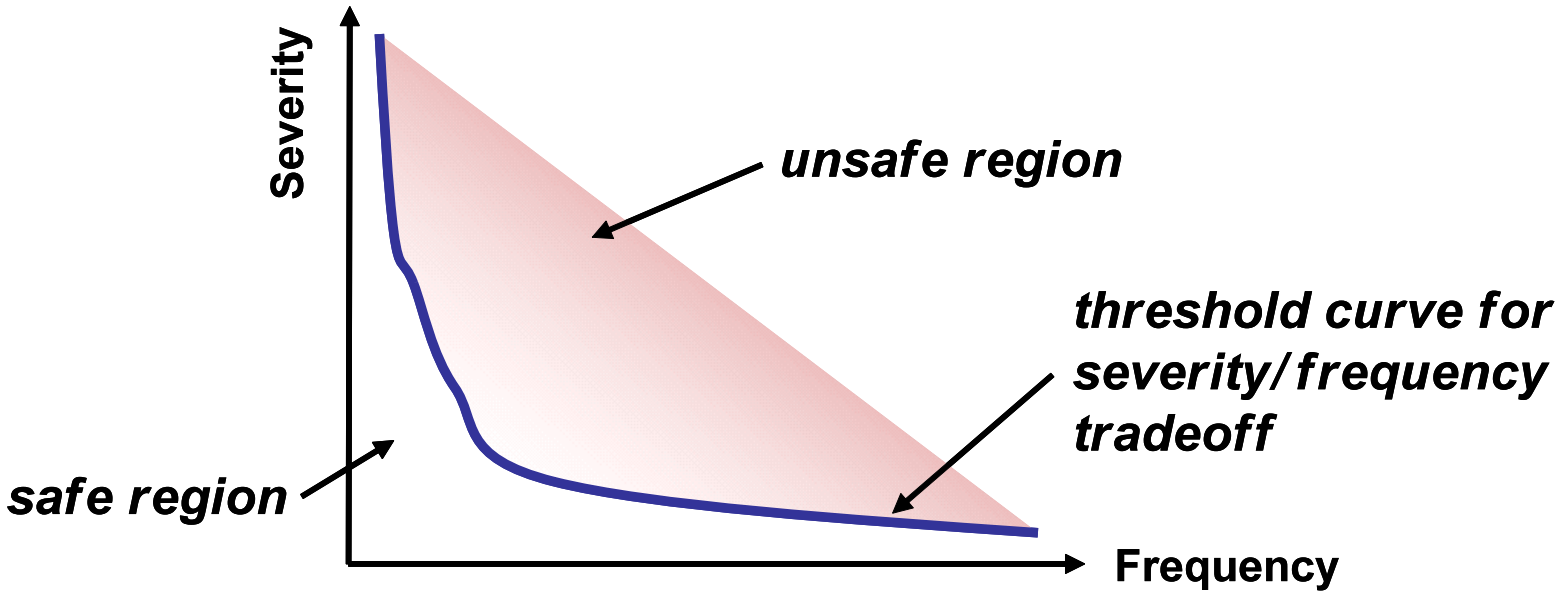
- the design of safety-related systems is driven by safety standards
- safety standards contain
  - rules and regulations for all design system
  - recommended guidelines for the development process
- safety standards cover all stages of the development process
  - specification
  - design
  - implementation
  - test
  - maintenance
- *objective of safety related design*
  - avoid unacceptable risk
  - assure functional safety

Functional safety

- safety*:* Freedom from unacceptable risk of physical injury or of damage to the health of people
- functional safety*:* refers to the safety of system functions
- risk is characterized by two properties
  - *frequency of hazardous events*
  - *severity of hazardous events*

Frequency and severity

- The idea: *frequency-severity tradeoff*



Functional safety – a short overview

- safety standards (IEC 61508, ISO 26262) classify systems according to frequency and severity of functional failures
- a safe system can handle faults without causing severe functional failures
- terminology

Safety standards - Overview

- **IEC 61508**
  - generic standard for safety-related systems
- **ISO 26262**
  - safety standard for automotive domain
- DO 178B, DO 254
  - safety standards for aerospace domain
- IEC 61511, IEC 62061
  - safety standards for factory automation domain
- EN 50126, EN 50128, EN 50129, EN 50159-1, EN 50159-2
  - safety standards for rail domain

IEC 61508 – Overview

- provides methods to assess the risk of functions
  - based on metrics of severity and frequency of failures
- introduction of safety in the lifecycle, which consists of
  - management of functional safety, e.g. enforcement of independent review processes of safety-related components
  - enforcement of verification and evaluation methods to assure functional safety
  - dedicated hardware and software development methods and processes
- further parts of IEC 61508
  - glossary
  - application examples and guidelines

IEC 61508 – Metrics

- reference standard that is used to derive other standards (e.g. ISO26262)
- metric: "Safety Integrity Level" – SIL
  - defines four degrees of safety: from 1 (lowest) to 4 (highest)
  - specification of maximum failure rates for each level

SIL	*Low demand mode: average probability of failure on demand*	*High demand or continuous mode: probability of dangerous failures per hour*
1	$> 10^{-2}$ to $< 10^{-1}$	$> 10^{-6}$ to $< 10^{-5}$
2	$> 10^{-3}$ to $< 10^{-2}$	$> 10^{-7}$ to $< 10^{-6}$
3	$> 10^{-4}$ to $< 10^{-3}$	$> 10^{-8}$ to $< 10^{-7}$
4	$> 10^{-5}$ to $< 10^{-4}$	$> 10^{-9}$ to $< 10^{-8}$

Reliability analysis and functional safety in IEC 61508

- *basic principle:* apply reliability analysis to verify that safety requirements are satisfied
  - *assumption:* required safety level is known a priori → hazard analysis and risk assessment not considered
- IEC 61508 does not directly support mixed criticality systems

  "An E/E/PE safety-related system will usually implement more than one safety function. If the safety integrity requirements for these safety functions differ, unless there is sufficient independence of implementation between them, the requirements applicable to the highest relevant safety integrity level shall apply to the entire E/E/PE safety-related system."
- reliability analysis can help to close this gap!
  - *more later*

Functional safety – ISO 26262

- ISO 26262 basically similar to IEC 61508
  - includes risk classification
  - defines development processes and method for saftey-critical automotive system
    - FMEA (failure mode and effect analysis), FTA (fault tree analysis)
- ISO 26262 defines ASIL 1-4 (automotive SIL) analogous to IEC 61508 SIL
- includes risk analysis and ASIL assessment process according to parameters severity, exposure and controllability
  - risk as a function of frequency f and severity S: R = F (f, S)
  - frequency as a function of exposure E and controllability C: f = E x C

Functional safety – Risk assessment matrix

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

note: **the class QM (Quality Management) denotes "no requirement" according to ISO 26262**

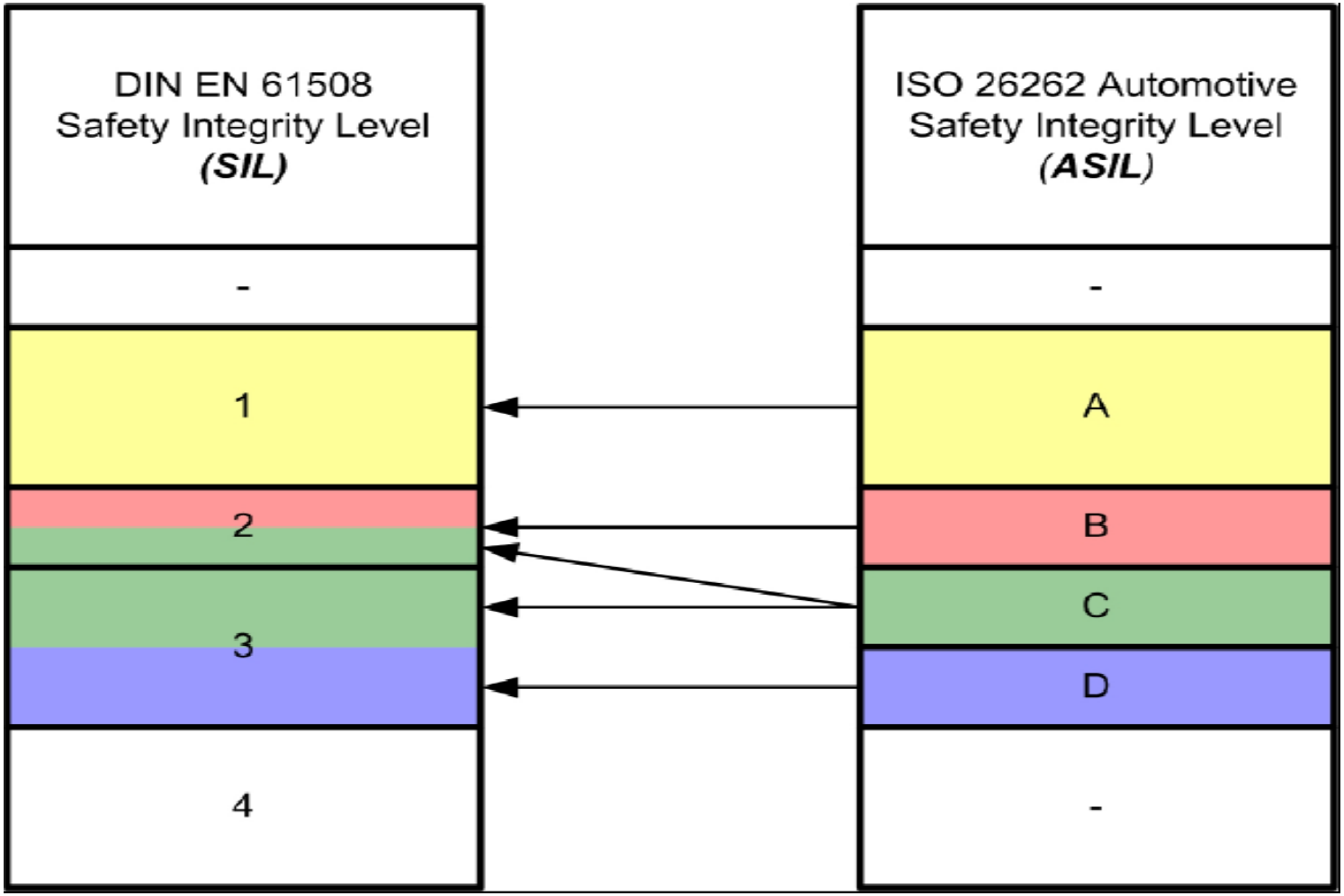
Functional safety – ISO 26262

- gap to IEC 61508: ISO 26262 provides no formal failure rate specification such as 61508
- however: approximate mapping is possible based on the term of "observable incident rate" introduced in ISO 26262

ASIL	Observable incident rate
D	$<10^{-8}$/h
C	$<10^{-7}$/h
B	$<10^{-7}$/h
A	$<10^{-6}$/h

- the observable incident rate is based on relevant field data
- basically observable incident rate is used for the proven in use argument
- ***"Proven in use argument is an alternate means of compliance with ISO26262 requirements that may be used in case of reuse of existing items or elements when field data is available."***

Result: SIL – ASIL mapping

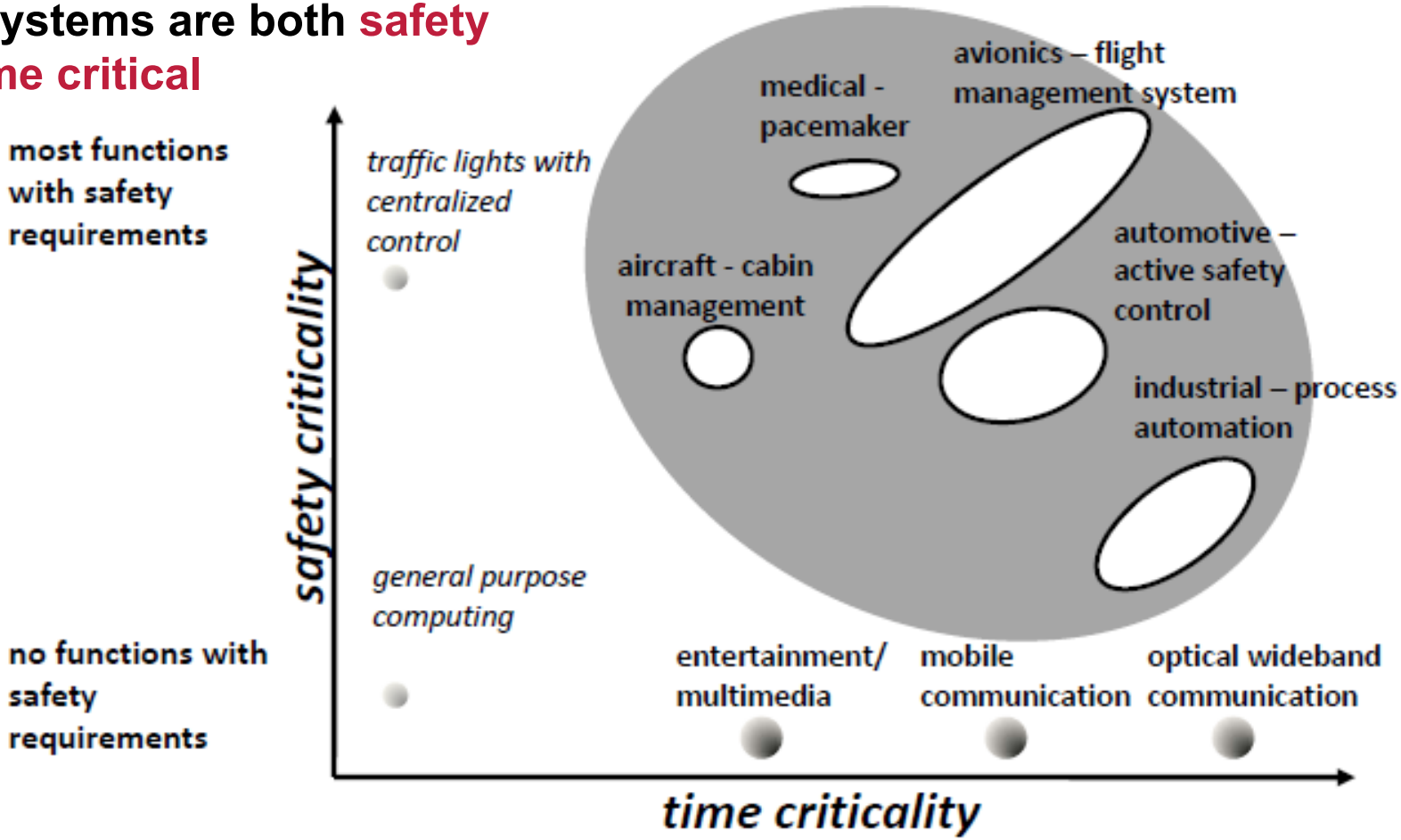


Embedded systems functional failures

- embedded system (ES) functional failures are not necessarily catastrophic
- effect depends on the importance of the failing function for the overall system
  - function criticality
- criticality depends on the overall system functionality
  - fail safe
    if the ES function fails there is a safe function backup or a safe system state that avoids severe consequences
    (mechanical steering, hydraulic brake, emergency stop)
    - ES is not critical but important for quality
  - fail operational (fault tolerant)
    the function continues based on system redundancy or turns to an error mode with reduced functionality (graceful degradation)
    - ES function is critical, but possibly only needs a specific function

Safety and time criticality - Reminder

- many safety critical systems have hard deadlines
  - such systems are both safety and time critical



Embedded system functional failures and timing

- ES functions have different criticality
  - depending on the overall system
- where timing is specified, it becomes part of the function criticality
  - ES timing failures are ES functional failures
- switching to error modes is time critical
  - switching needs hard deadlines to guarantee overall system function

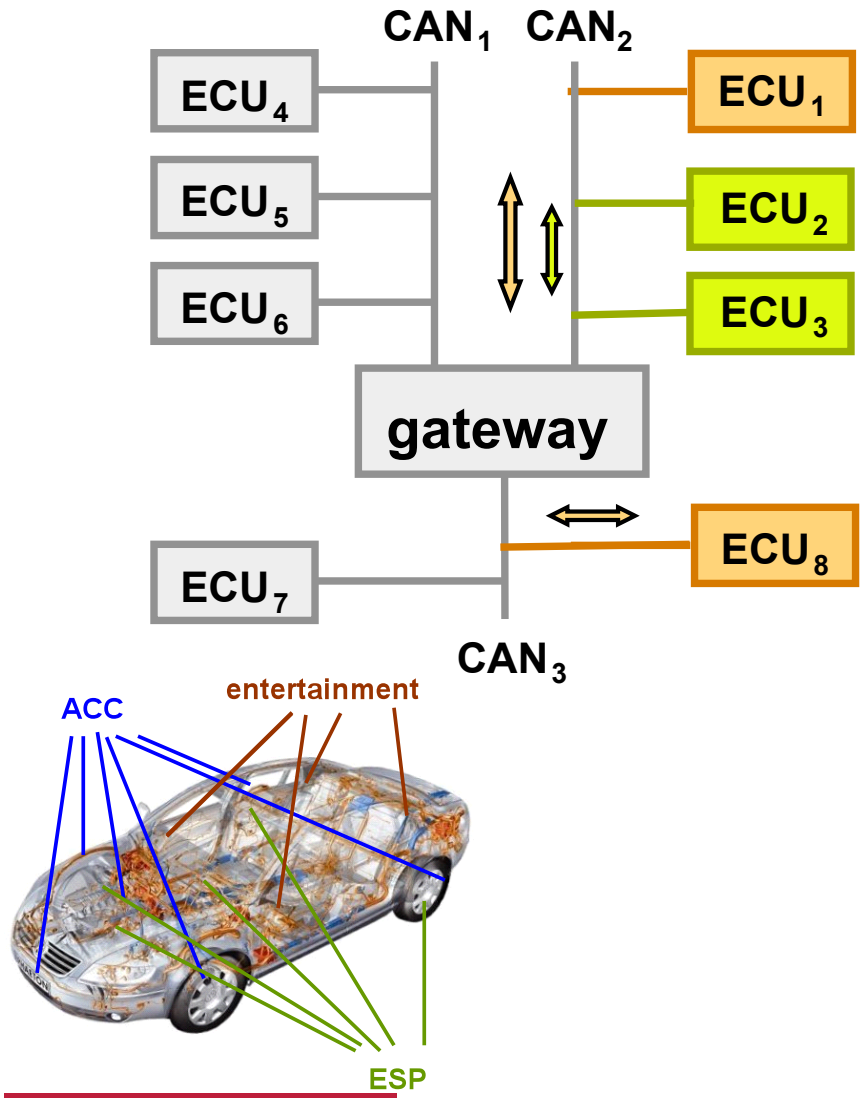
Overview

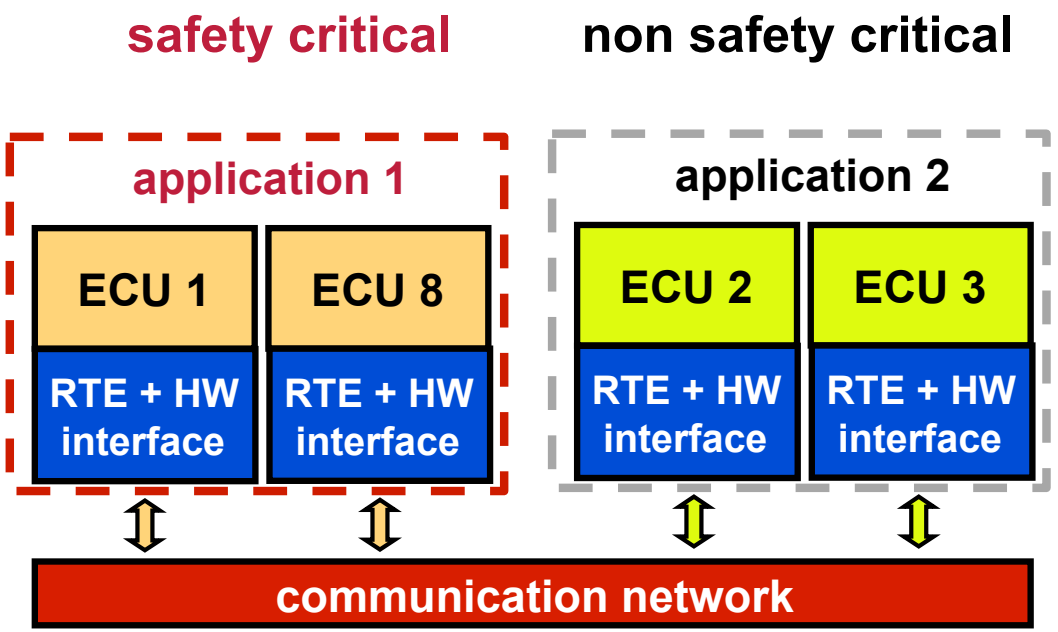
- motivation
- safety critical embedded system design
- **networks and multi-core systems for mixed time and safety critical applications**
- reliable systems for higher safety requirements
- example projects
- conclusion

Safety challenges in ES integration

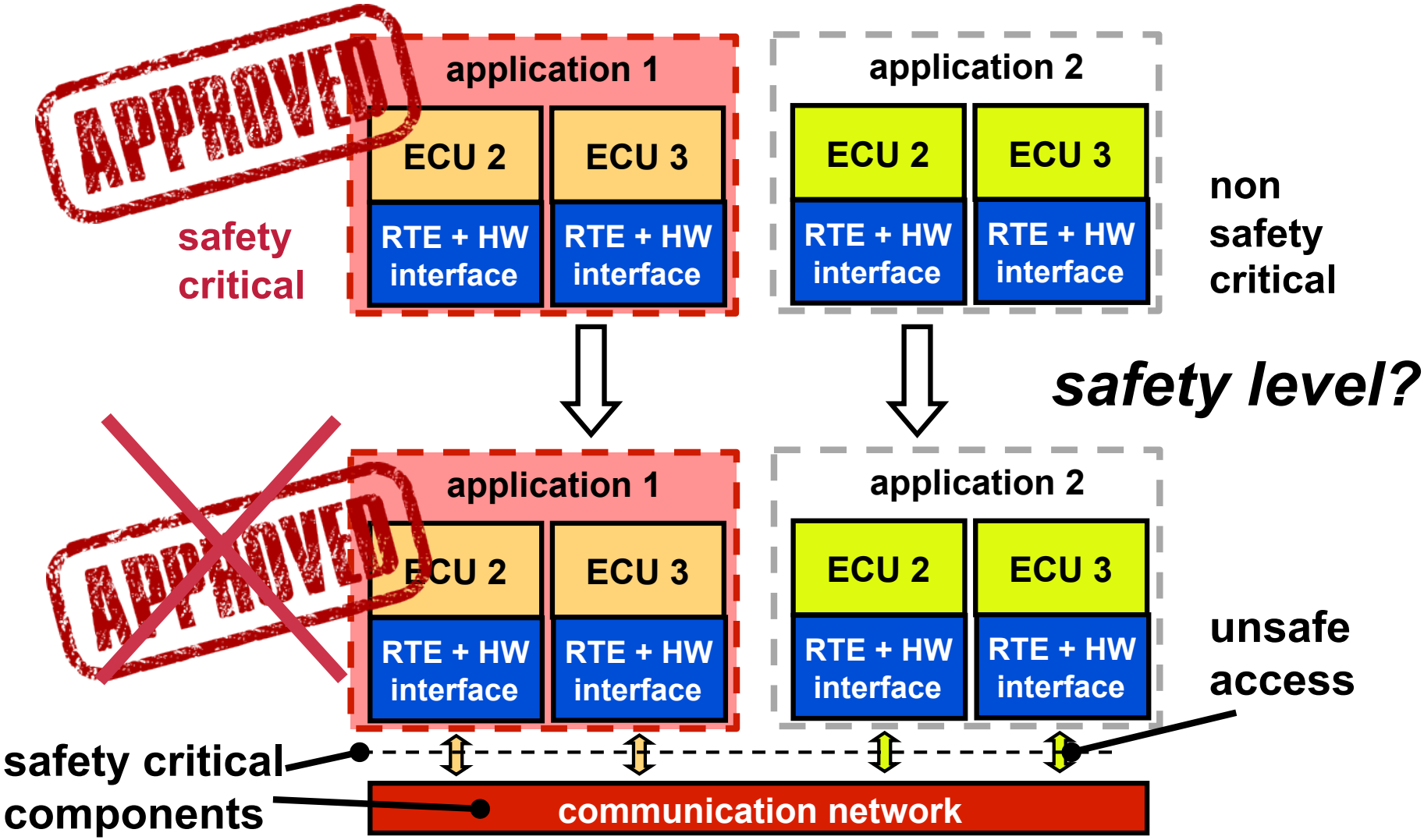
- sharing resources is hard to avoide in cost efficient systems
  - shared (open) network
  - shared on-chip network, shared memories, …
- is it possible to integrate several subsystems and avoid interference?
  - this would be important for mixed criticality systems
    - *non-critical parts are less verified and not designed for worst case*
  - would reduce verification/certification/integration cost
- standards require separation in case of shared resources
  - Reminder (IEC 61508)
    "… If the safety integrity requirements for these safety functions differ, unless there is sufficient independence of implementation between them, the requirements applicable to the highest relevant safety integrity level shall apply to the entire E/E/PE safety-related system."

Reminder – Automotive network



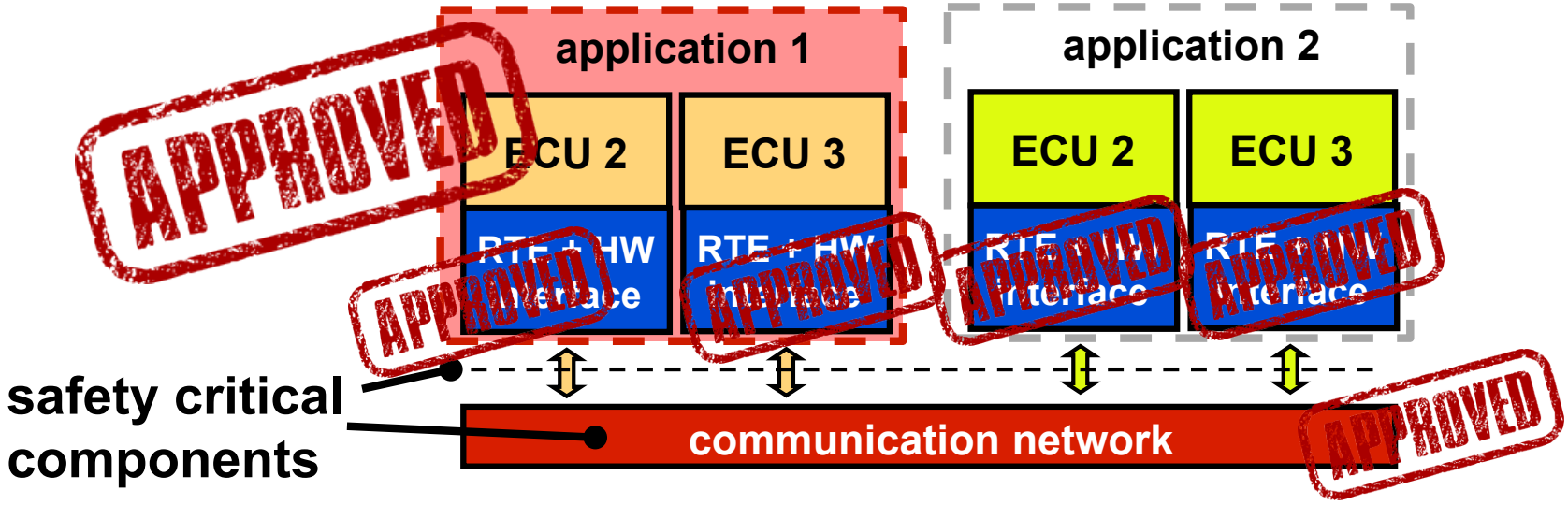


Critical application using network - Consequence

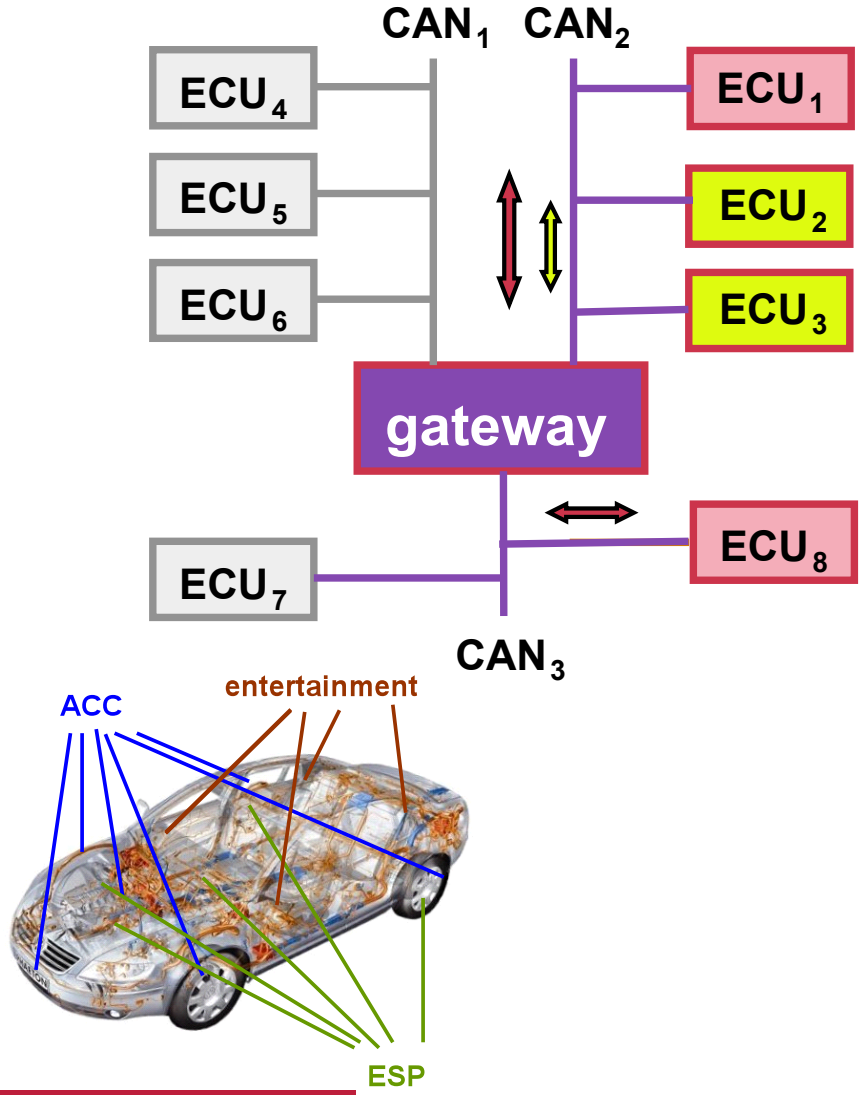


Separation - Principle

- partitioning into certified/qualified core components that control the resources used for any of the critical applications
  - basic software incl. RTE
  - communication
  - shared resources used for critical applications



Automotive network – Affected system parts



- application of safety standard affects large part of the system
  - single criticality on safety critical ECUs
  - mixed criticality on other parts

Example Automotive - Communication

- CAN
  - multi master, non synchronized
  - static priority non preemptive (SPNP)
  - needs formal analysis to guarantee arrival of critical messages
  - error handling protocol
- FlexRay
  - fixed sequence of static segments with TDMA protocol and dynamic priority assigned segments – cyclo-static repetition
  - time synchronized, multi master
  - guaranteed resource share for each communication channel
- gateways
  - proprietary solutions

Separation on CAN

- assumptions
  - all senders adhere to their priority i (message id)
  - no two messages of the same type or priority are on the bus
    - requires that latest deadline is at end of period
  - buses are not overloaded (U < 100%), messages don‘t miss deadlines
  - then (simplified):

$$R_i = C_i + \sum_{j \in hp(i)} C_j \cdot \left\lceil \frac{R_i}{T_j} \right\rceil + \max_{k \notin hp(i)} (C_k)$$

$$\forall i : R_i \leq T_i$$

- R_i response time message i; T_i min. period
- C_i execution/frame transmission time
- hp(i) higher priority messages

  - for CAN: all C_i equal (constant frame size)

⇒ worst case response times only influenced by higher priority messages

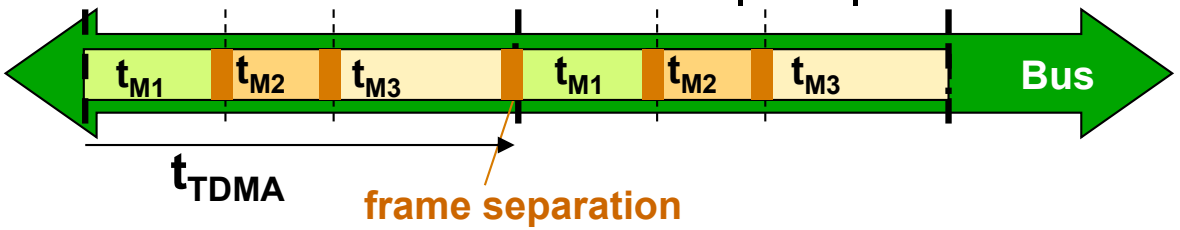
⇒ critical communication independent of other communiation if given higher priority *(no RMS ⇒ non optimal scheduling)*

Separation on FlexRay

- assumptions
  - all senders adhere to the TDMA schedule
  - all clocks are synchronized
  - messages don‘t miss deadlines if TDMA schedule is regarded
  - then (simplified) for the static segment:

$$R_i = C_i + (t_{TDMA} - t_{Mi}) \times \left\lceil \frac{C_i}{t_{Mi}} \right\rceil$$

R_i response time task i; t_{Mi} time
C_i execution/frame transmission time
t_{TDMA} TDMA cycle time



⇒ worst case response times not influenced by any other message

⇒ complete separation of logic channels

Separation on CAN and FlexRay

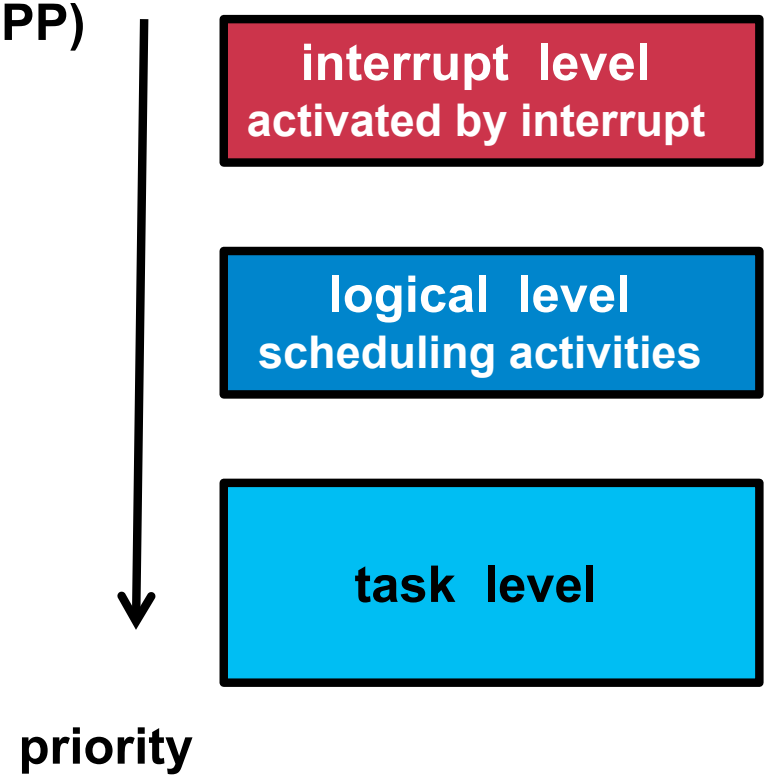
- both bus protocols support separation of critical from non critical messages
  - FlexRay static segment enables separation of all messages, CAN provides an asymmetric separation
  - all senders must give guarantees
    - CAN: keep message priorities (hardware based conflict resolution)
    - FleyRay: adhere to global time and TDMA schedule

Separation in processing units

- uses same scheduling principles as communication
  - static priority driven scheduling – automotive: OSEK/VDX and AUTOSAR
  - TDMA – avionics: ARINC 653
  - main principles used in (mixed-critical) practice
    - others proposed

Scheduling in OSEK/VDX and AUTOSAR

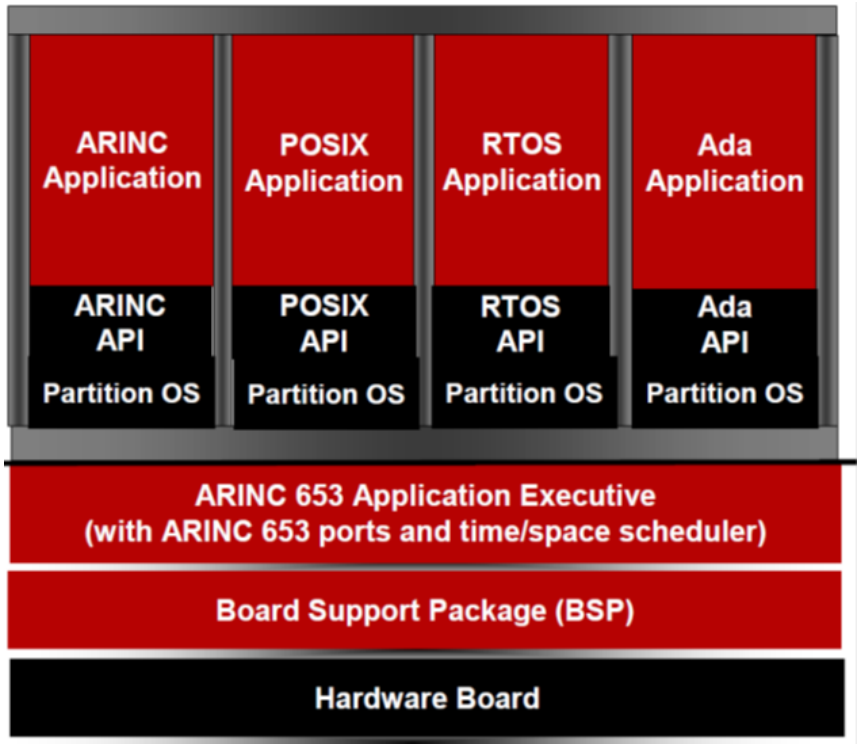
- ECUs typically uses OSEK/VDX RTE
  - static priority preemptive scheduling (SPP)
    - can be restricted to preemption points
  - three priority blocks
    - interrupt – scheduling – task level
  - task level w. periodic task execution
    - Rate Monotonic scheduling
    - offsets for load bounding
  - PCP protocol to bound blocking by resource arbitration
  - no standard memory access protection



source: OSEK/VDX standard V2.2.3

Separation in processing units

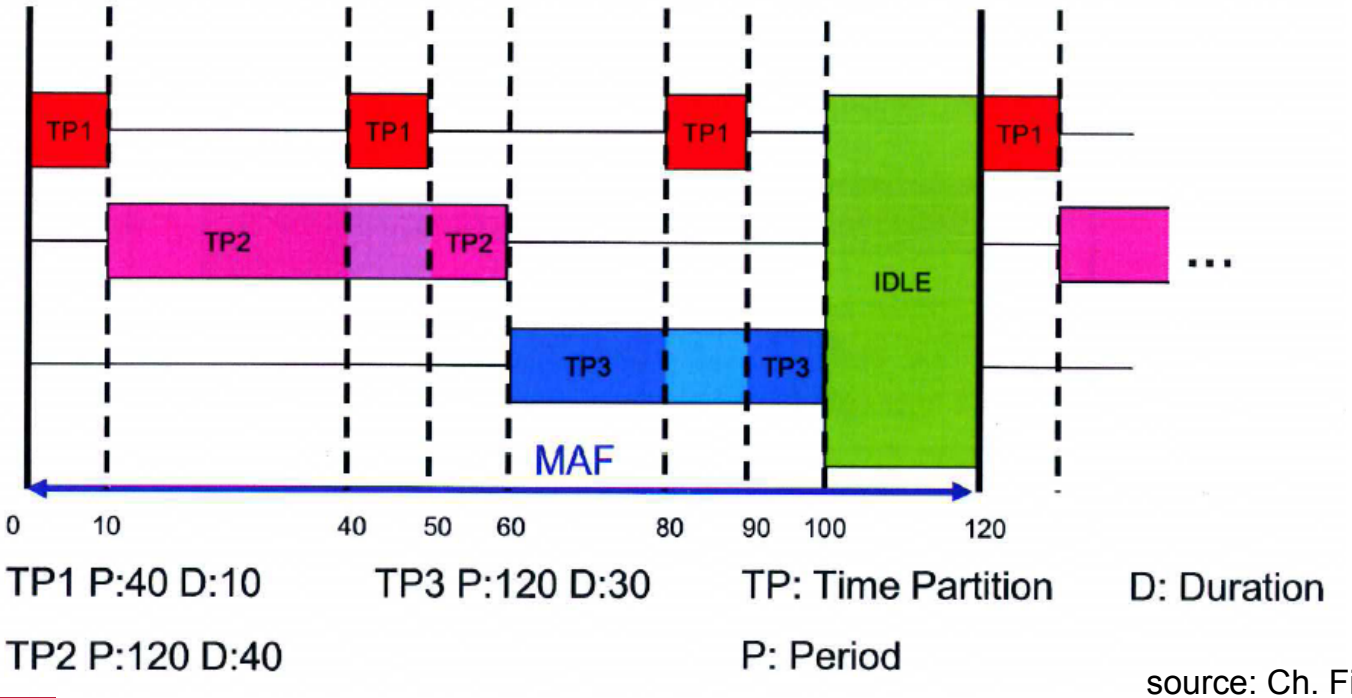
- **ARINC 653 – Integrated Modular Architecture IMA**
  - **several systems with separate OS implemented on one CPU**
  - **separated address and memory spaces (requires MMU)**



Source: Wind River, 2008

Separation in processing units

- **ARINC 653**
  - **partitions are assigned to time windows TPi iterating over a major Time Window MAF**
  - **execution can exceed single time window**
  - **supports scheduling hierarchies**



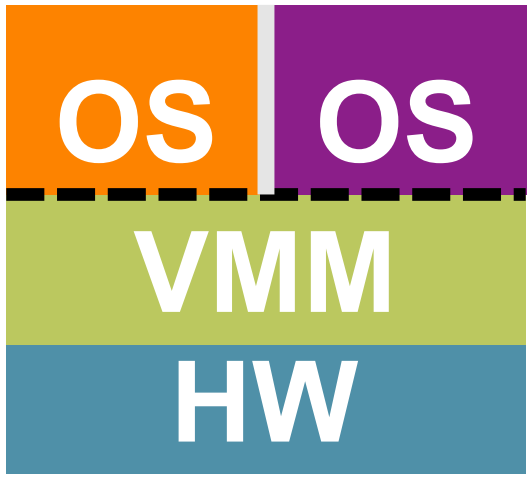
source: Ch. Ficek, Symtavision

Separation in RTOSs

- timing separation similar to communication examples
- separation of memory and device usage requires access control
  - approaches: virtualization or memory (address) protection
- consequence
  - all RTOS mechanisms needed for separation must be subject to the highest level of criticality in the system

Virtualization - principle

- decoupling of virtual and physical resources
- a virtual machine monitor (VMM) administers physical resources such as processors (CPU time), memory, peripherals
- In classical desktop/server virtualization the VMM splits the physical computing platform into independent virtual platforms
- some use cases
  - consolidation of services on one physical platform
  - running different/legacy Oses on the same platform
  - containment of services in its own virtual platform
  - architectural abstraction: a virtual machine can easily be migrated

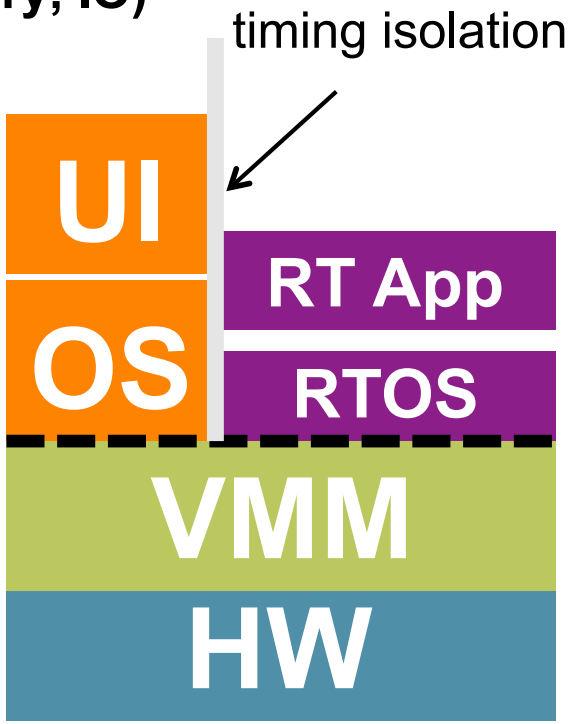


Virtualization techniques

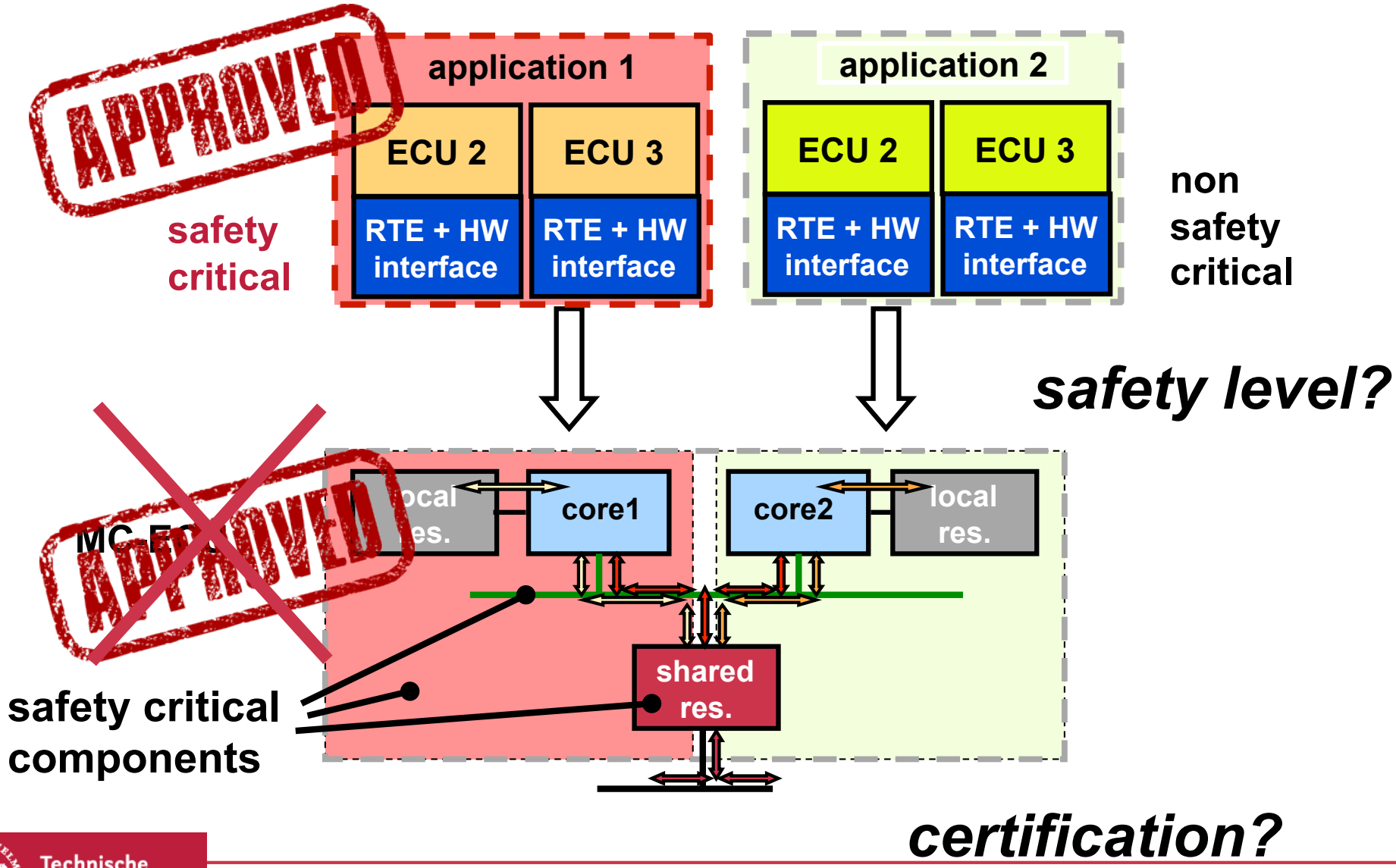
- paravirtualization
  - explicit API interface between guest OS and VMM
    - example XEN: guest OS calls HYPERVISOR_mmu_update → Xen updates the MMU
    - (proprietary) guest OS must be ported to the VMM API
    - hardware emulation required → fast
- full/hardware virtualization
  - relies on Hardware Support (Intel VT/ AMD Pacifica)
  - VMM emulates the standard hardware (e.g. chipset, ethernet)
  - when an IOMMU is present: phyical peripherals can be mapped into the guest OS
  - slower than paravirtualization, but supports legacy/propriatary OSs

Virtualization in embedded systems

- integration of mixed-critical applications supported by virtualization
- challenges
  - VMM introduces additional timing latency
  - Shared resources on multi-core architectures (memory, IO)
  - Additional cache misses and additional IRQ sources
- see ARINC653

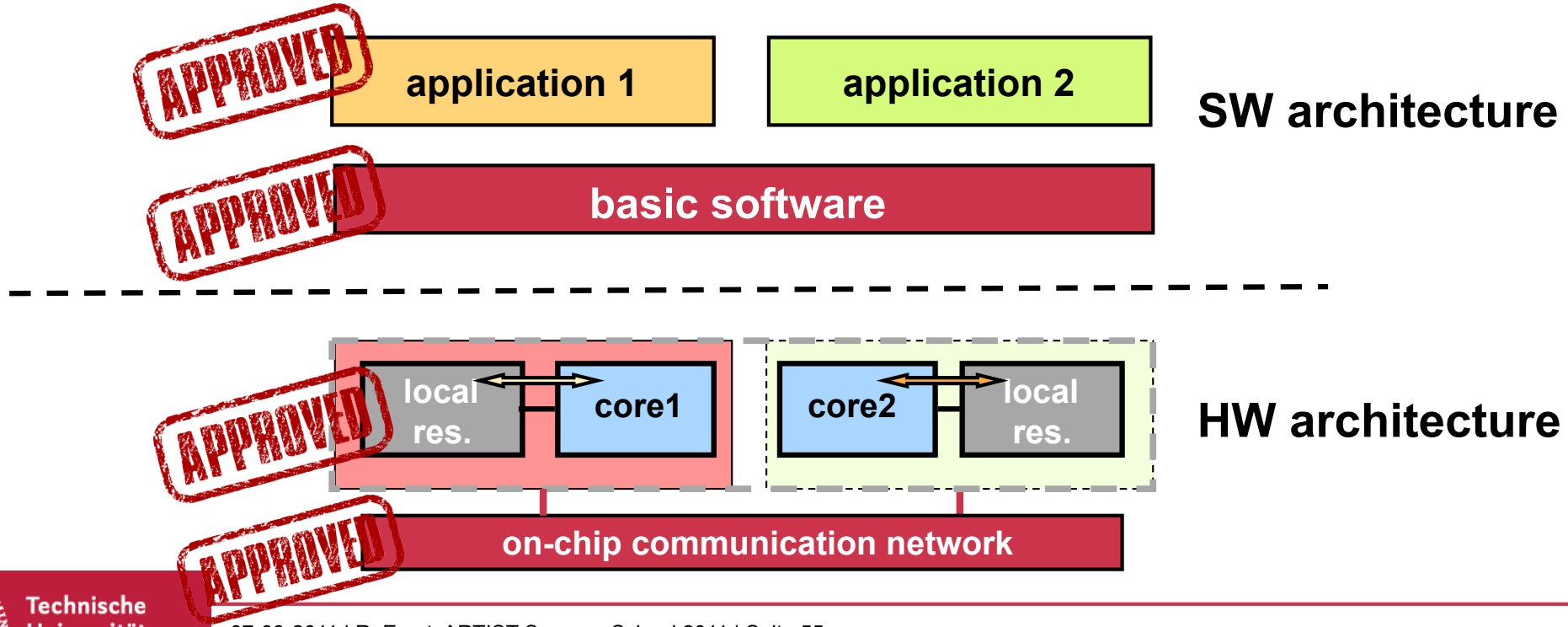


Mixed critical applications in multi-core architectures



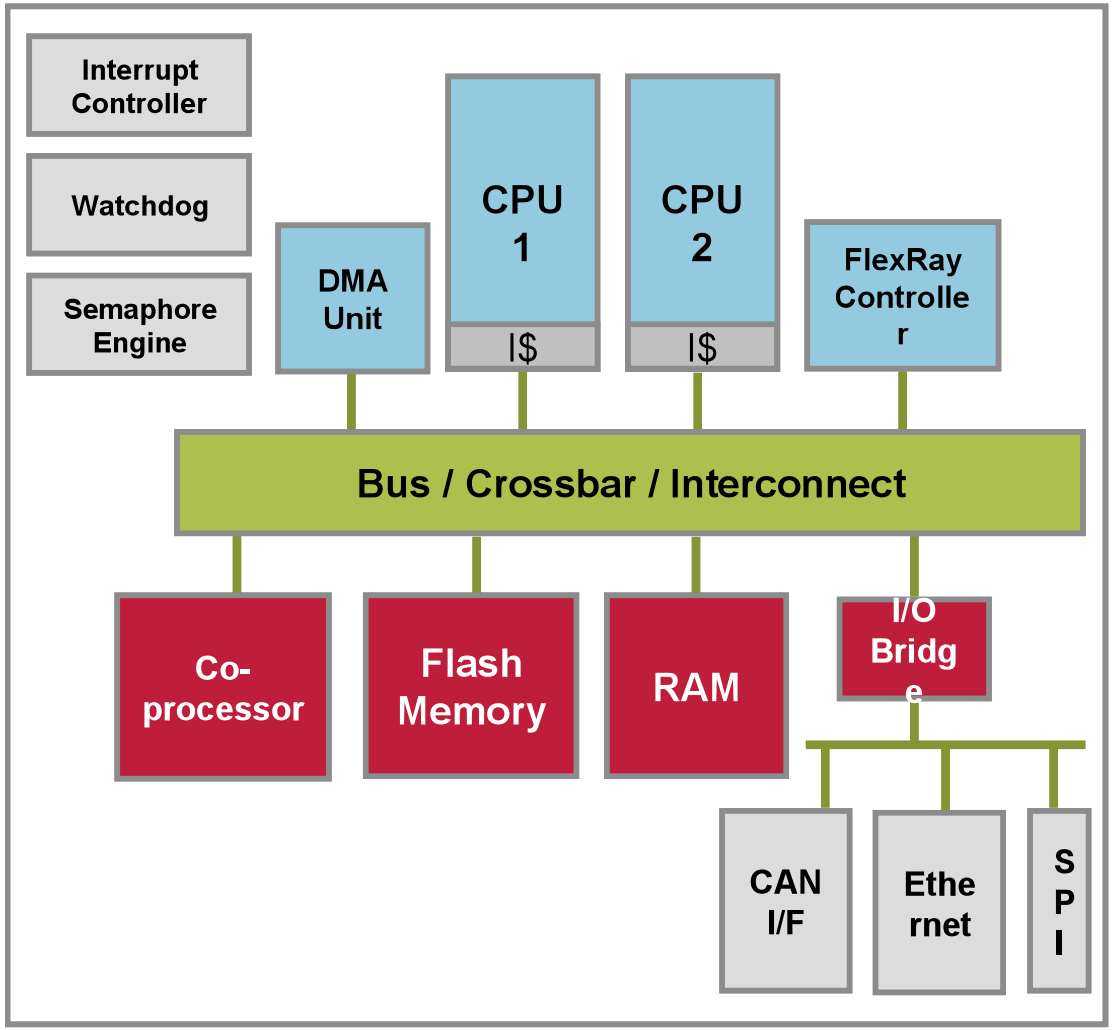
Multi-core separation - Principle

- partitioning into certified/qualified core components that control the resources used for any of the critical applications
  - basic software incl. RTE
  - communication
  - shared resources used for critical applications



Separation in multi-core architecture

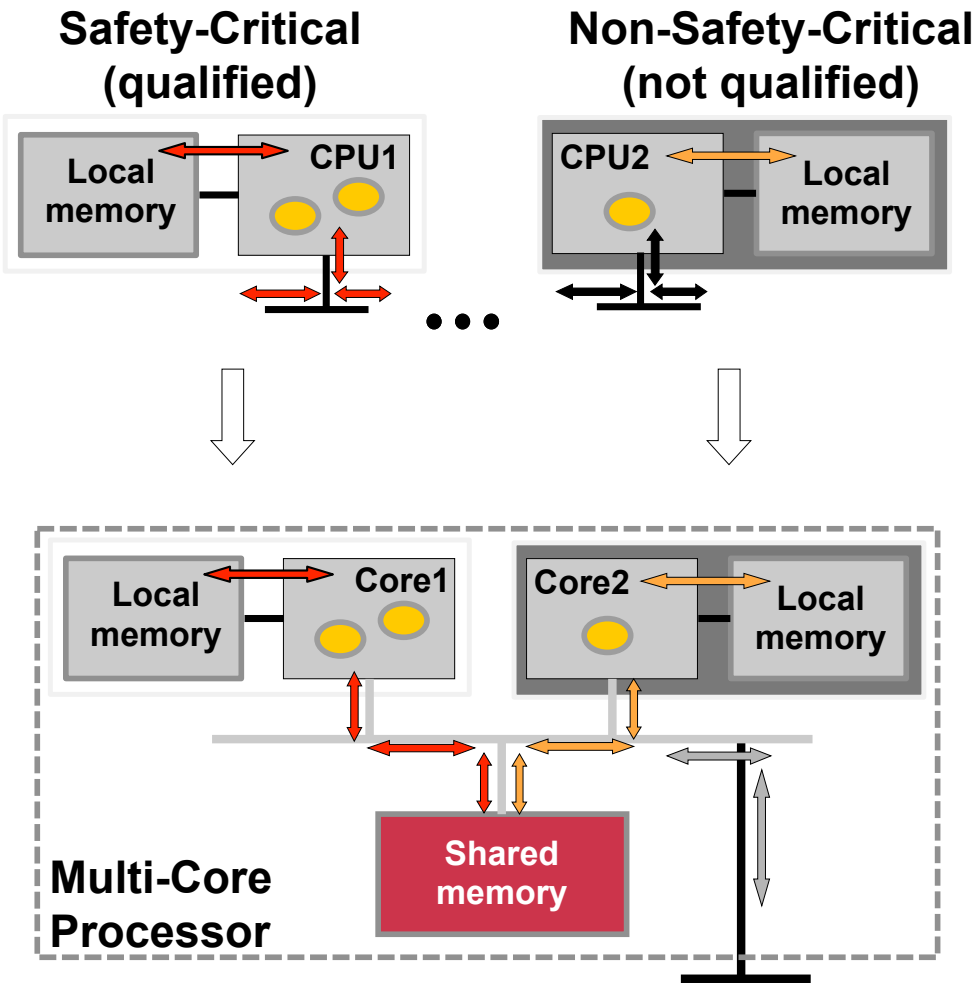
- standard approach for separation - isolation
  - separate address spaces and cores
    - possibly controlled by hypervisor (virtualization)
  - only allow event and data flow from higher criticality to lower criticality (safety requirement)
- *is this sufficient?*



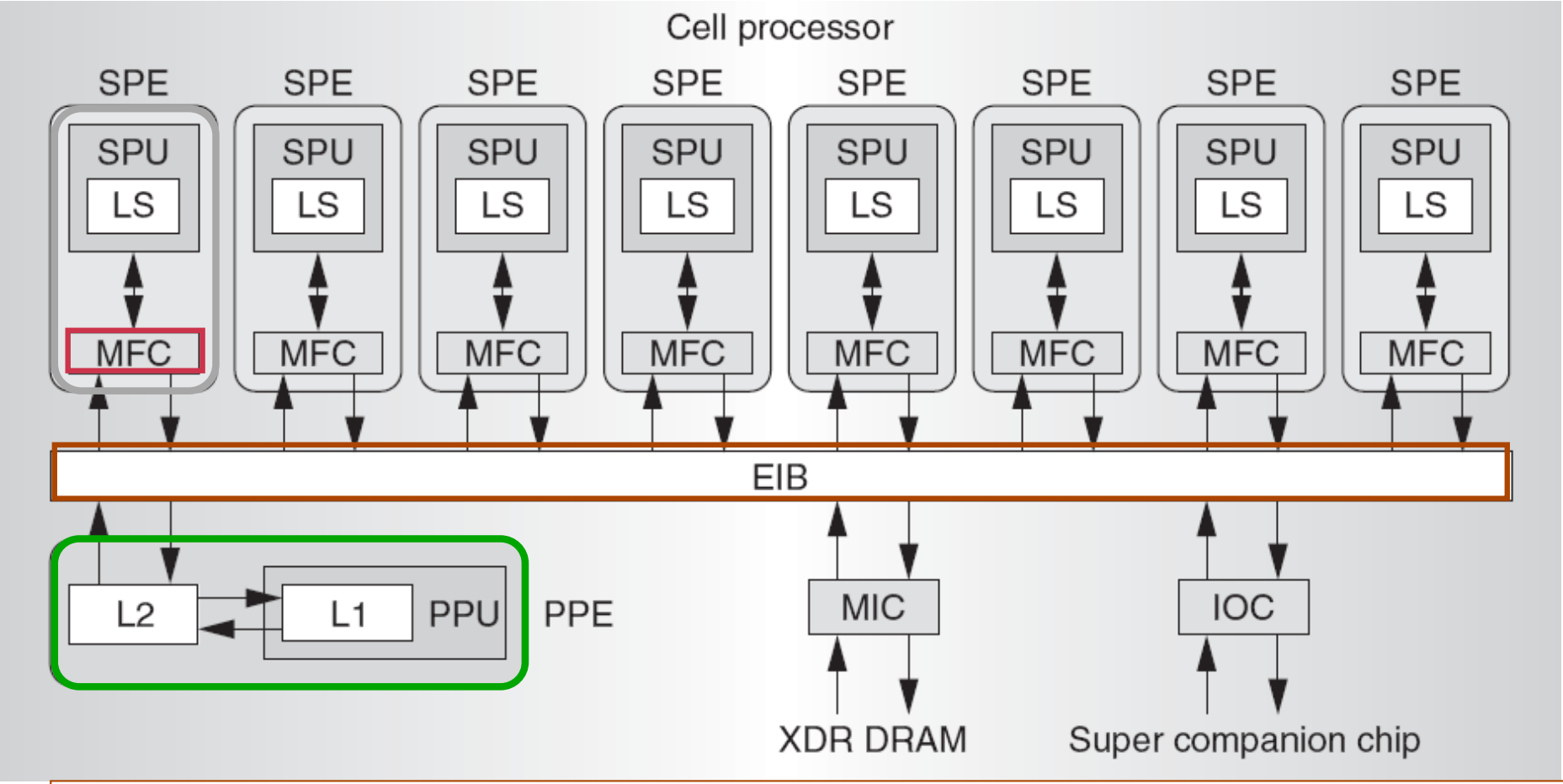
Multi-core example: Freescale MP5565

Multi-core example continued

- assumption
  - two cores integrating applications with different criticality levels
  - isolated address space
    - CPU2 cannot interfere with CPU1 data
  - Independent core scheduling
  - tasks access local and global shared resources (e.g. shared SRAM)
- consequence
  - functions are isolated
  - *but is timing isolated, as well?*



Example: Multi-core Cell Broadband Engine

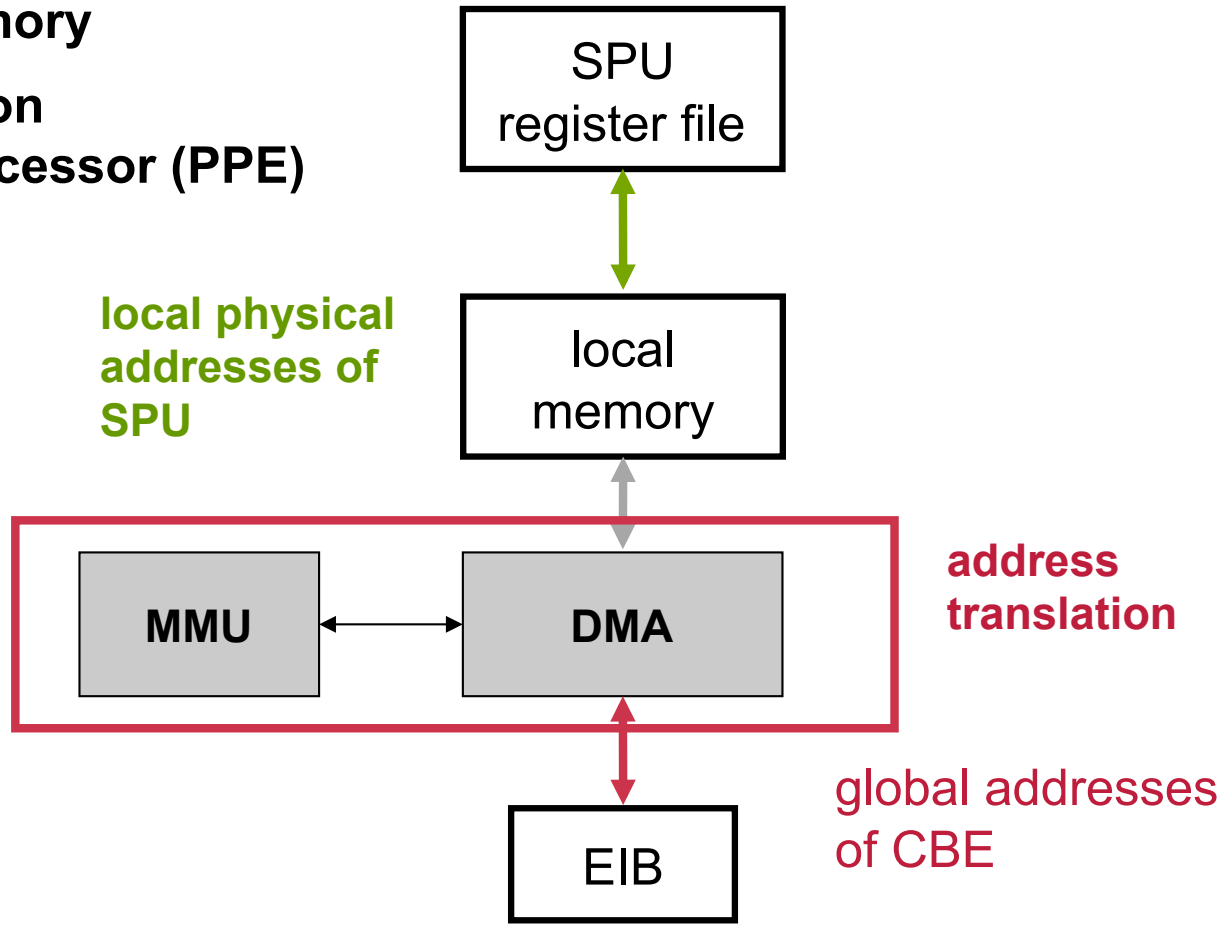


PPE: 64 bit Power (5) processor with 2 level cache architecture as main controller
SPE: Synergistic Processor Element – specialized processor
MFC: Memory Flow Control (DMA)
EIB: Element Interconnect Bus- high speed ring bus
MIC: Memory Interface Controller; IOC: IO Controller

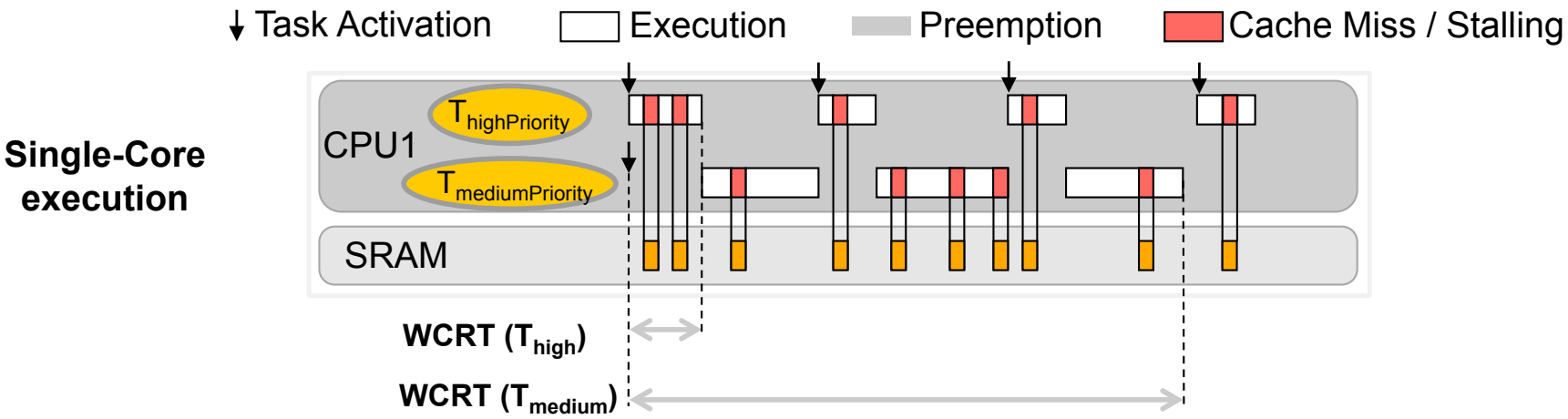
Source: IEEE Micro

CBE memory model – Local and global memory maps

- isolation of address space
  - SPEs work on local memory
  - global address translation controlled by global processor (PPE)
- *but: all cores share the same bus and external memory*
  - mutual timing influence (complicated)



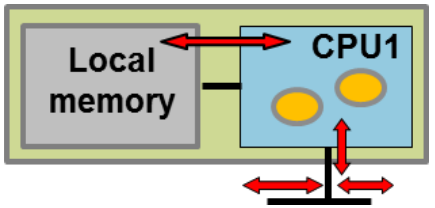
A closer look at timing: Single-core execution



- on CPU1
  - when a task is waiting for the SRAM the processor is stalled ("micro lock")
  - $T_{highPriority}$ and $T_{mediumPriority}$ initiate requests for the SRAM and have to wait for the required data

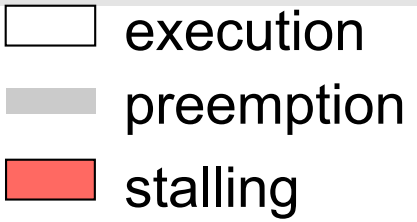
    → causes additional delays on the execution of other local tasks

Safety-Critical (qualified)

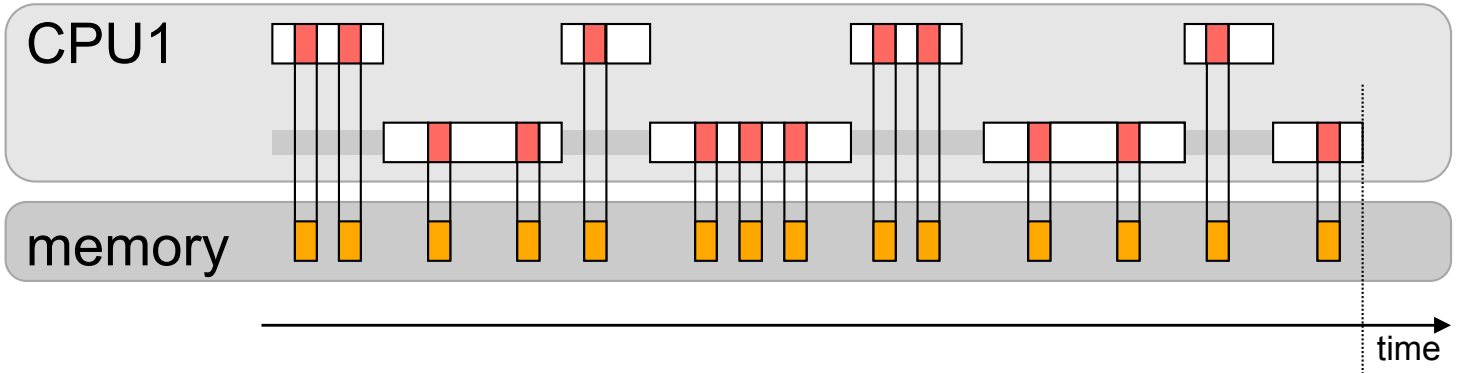


Application timing dependencies in multi-core

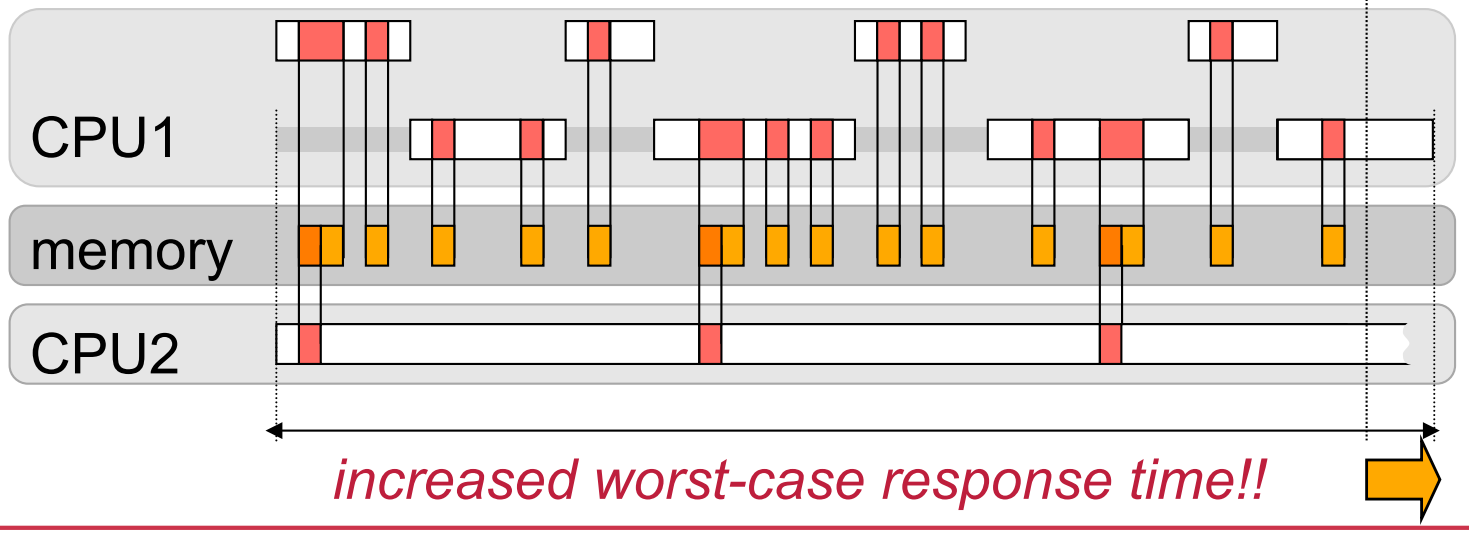
competing accesses to shared resources challenge worst-case execution time assumptions!



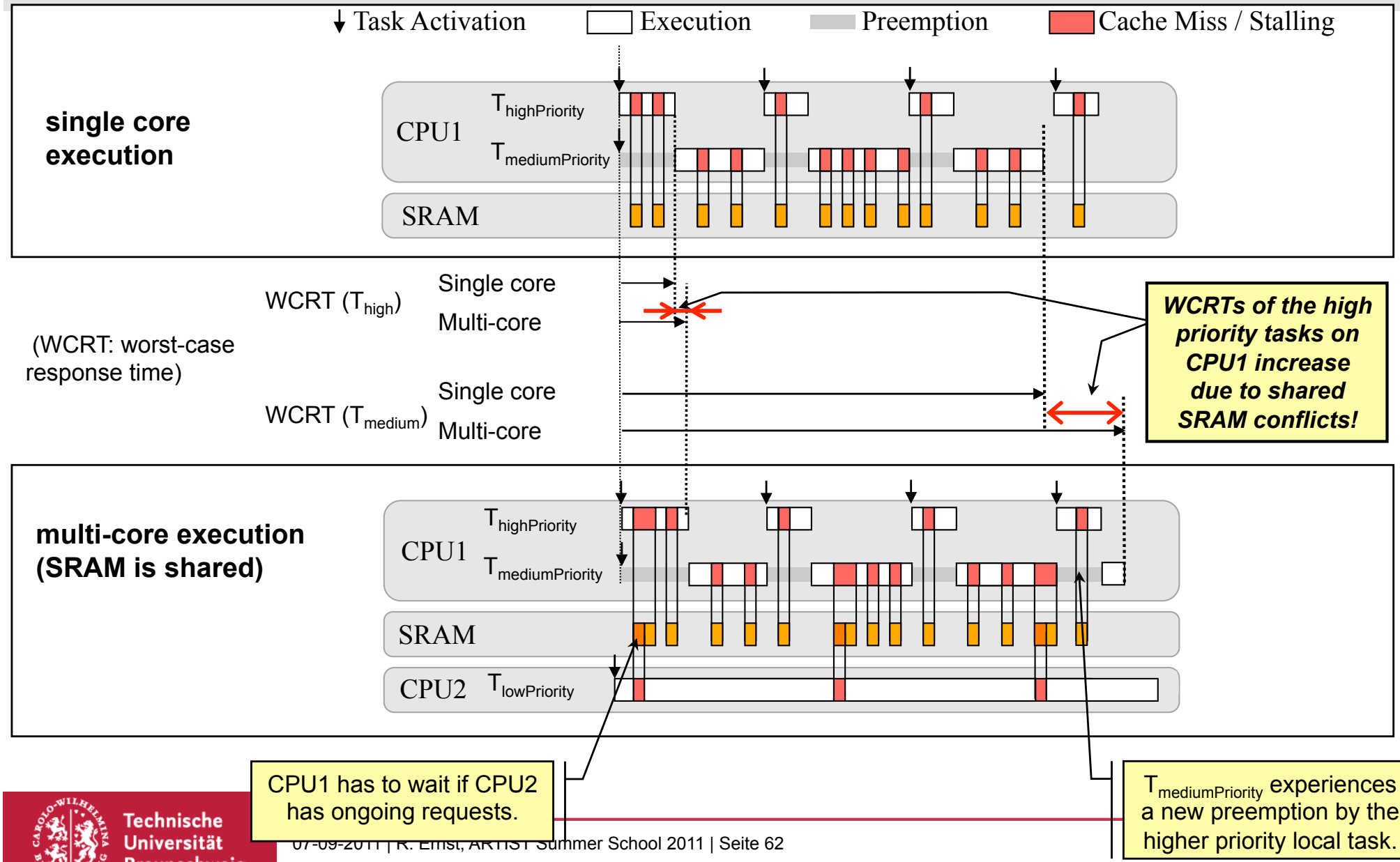
single core execution (with stalling)



multi-core execution



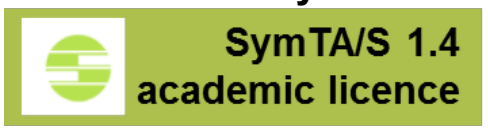
Competition for shared resources – a closer look



Example - System parameters

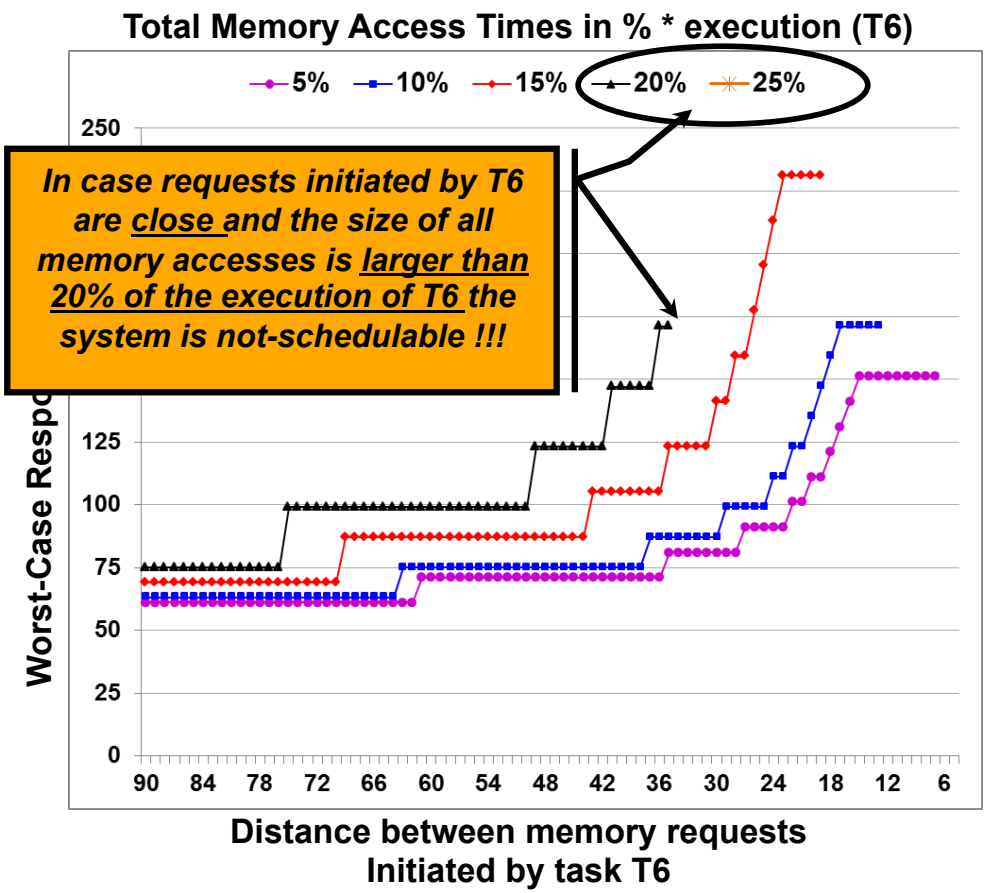
Scheduling	Mapping	Task	Priority P(1)>P(3)>P(5)>P(6)	Activation Period	WCET	# Memory Accesses per execution	Memory Arbitration
SPP	Core 1	T1	1	250	50	5	FCFS
	Core 1	T3	3	800	360	12	
SPP	Core 2	T5	5	1500	500	5	
	Core 2 (after update)	T6	6	10000	1200	10	

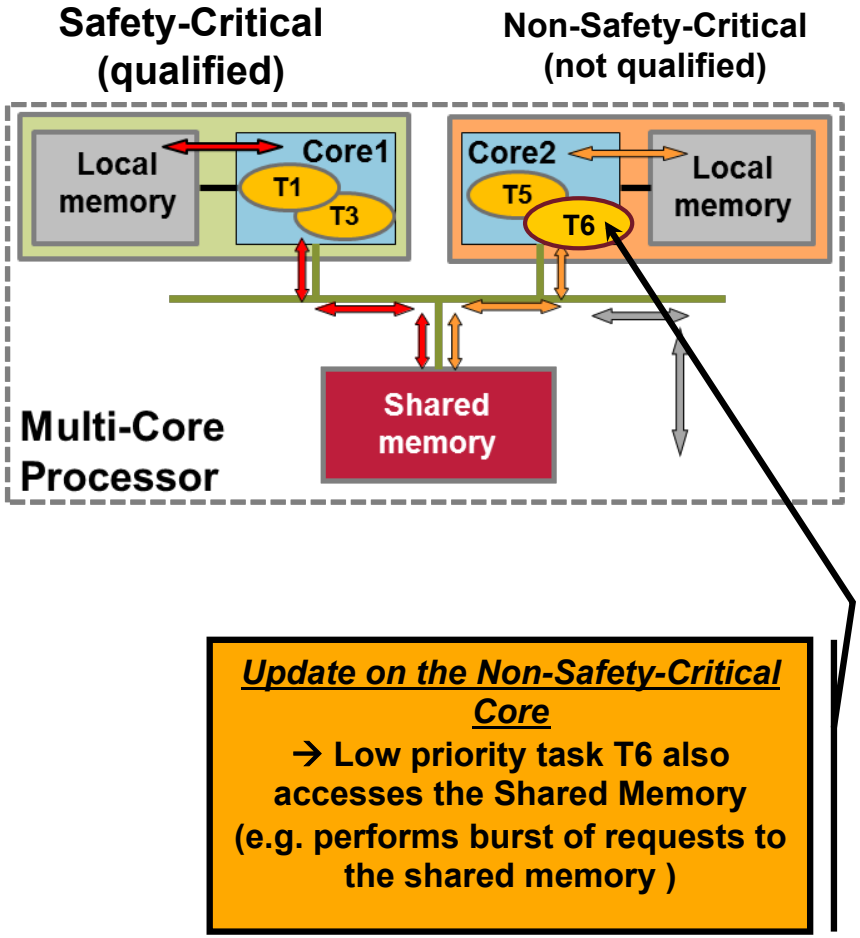
Modeled and analyzed with:



Integrate safety critical and non safety critical cores – SW update

- update on Core 2 challenges timing on Core1





Timing hazards when using shared resources 1/2

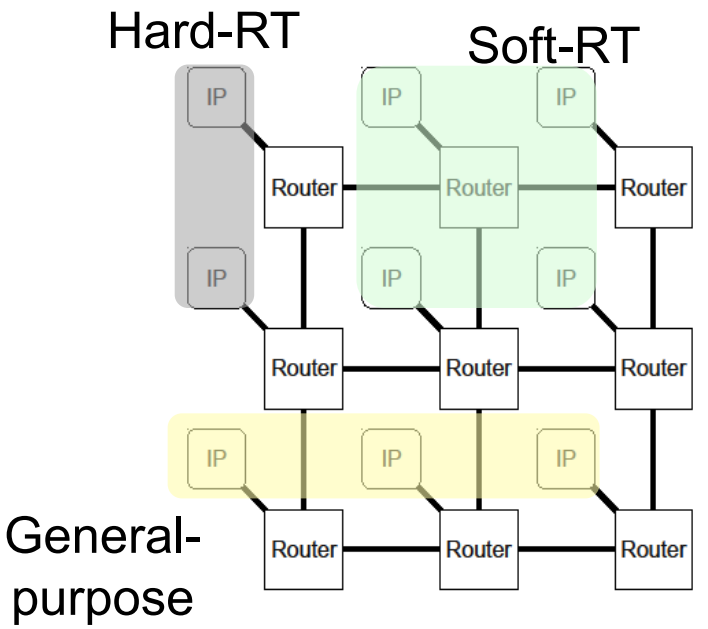
- common use of the SRAM among cores – observed effects
  - SRAM accesses by low priority tasks on one CPU may slow down other tasks on another CPU
    - affects low and high priority tasks likewise
    - counters priority assignment on the cores - priority inversion
  - WCRT may even increase super-linearly due to additional preemptions (shown for $T_{\text{medium priority}}$ in the example)
  - as CPUs are stalled when tasks are waiting for the shared memory the load on these CPUs will increase
  - problem demonstrated for accesses to shared SRAM, but the same reasoning applies to semaphore protected critical sections and other shared resources

Timing hazards when using shared resources 2/2

- *example shows high potential impact of non-critical on safety critical task* **despite high task priority and isolation**
  - **dangerous if software bugs in non-critical system with lower verification requirements (e.g. infinite loop w. memory access)**
  - **requires physical separation or (re-)certification including non-critical part**
    - **extra cost!**
- **Note:** ***Virtualization alone does NOT help!***

Controlled separation for many-core systems

- separation must include NoC and on-chip memories
- suggested approaches typically use strict resource separation
  - example: NXP Aetheral NoC , …
  - challenge: efficiency (performance loss)
- possible improvement
  - budgeting
  - channel separation
  - arbitration control for service guarantees

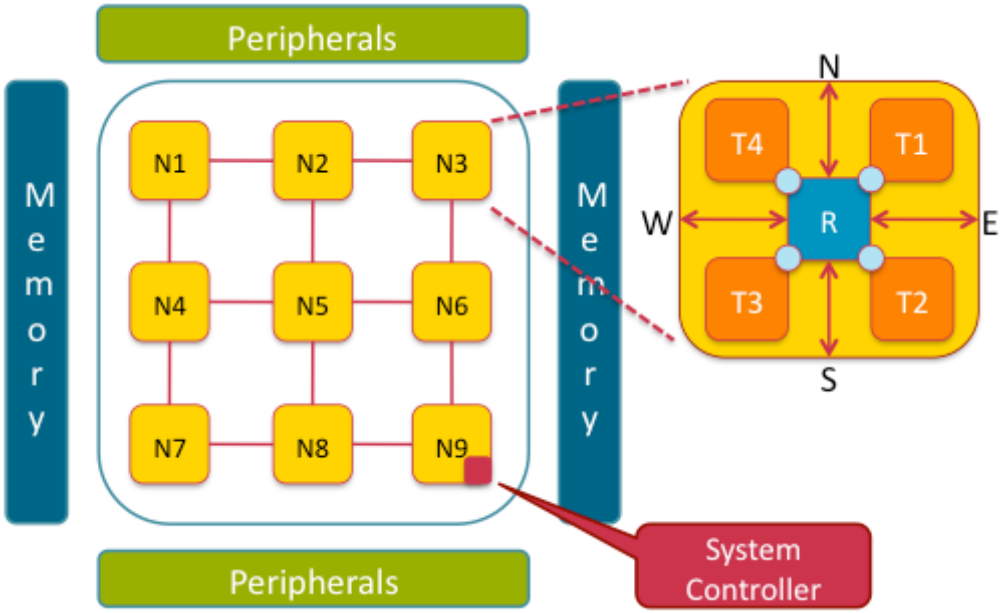


The missing links

- efficient separation of critical NoC traffic with minimized negative effect on non-critical traffic
  - avoid crossbar to enable many-core ICs
- include resource access protection
  - avoid resource overutilization by non critical applications
- example: IDAMC

Integrated Dependable Architecture for Many Cores (IDAMC)

- general purpose system with support for mixed-criticality
  - safety-critical real-time
    - timing guarantees
  - best-effort, e.g. office, games,
    - latency sensitive
- 4-64 nodes
  - mesh NoC with QoS
  - up to four tiles per Node
- hardware mechanisms for
- virtualization at NoC-Level
- monitoring (timing and power)
- on-chip data transport, c2c communication



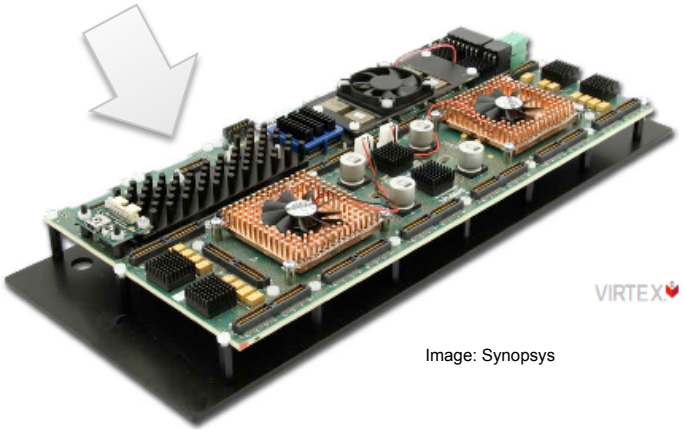
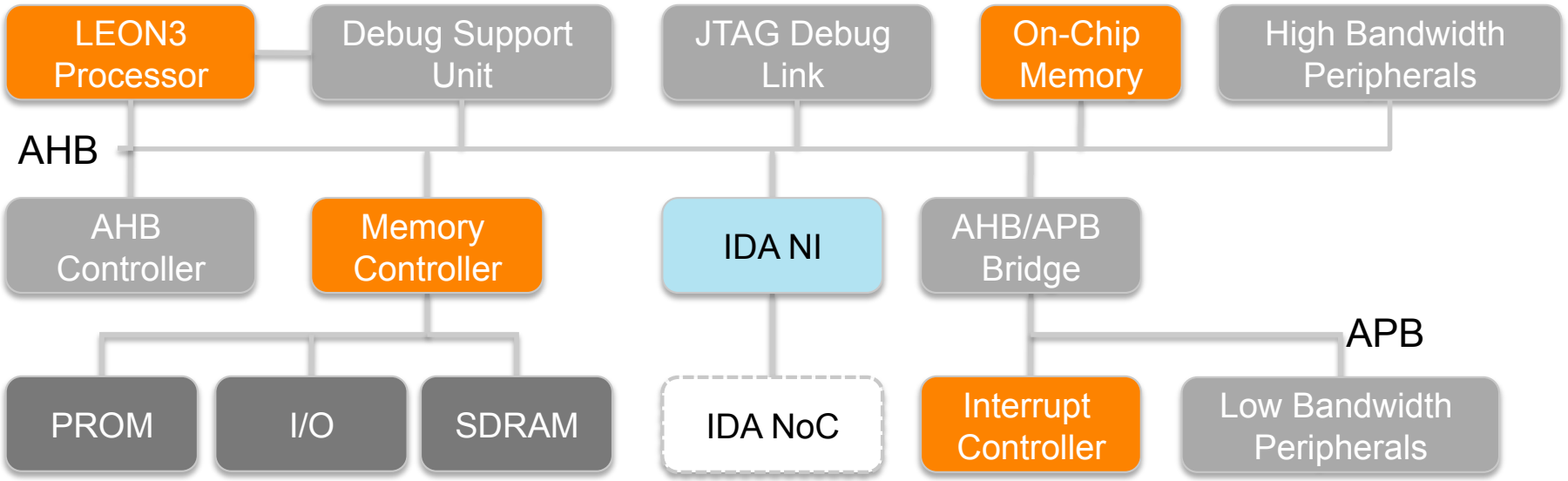


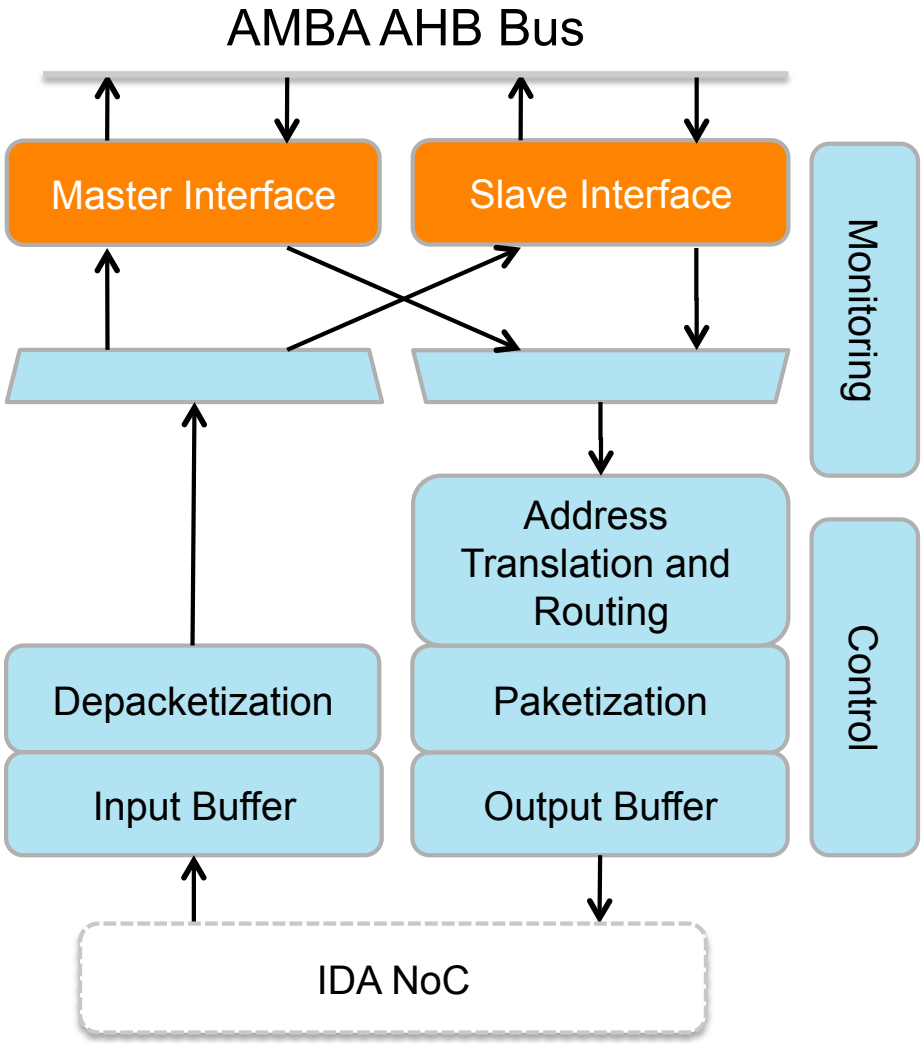
Image: Synopsys

Tiles

- each tile is a complete system
  - AMBA bus
  - LEON3 CPU and/or memory, peripherals
- network interface (NI) connects to NoC



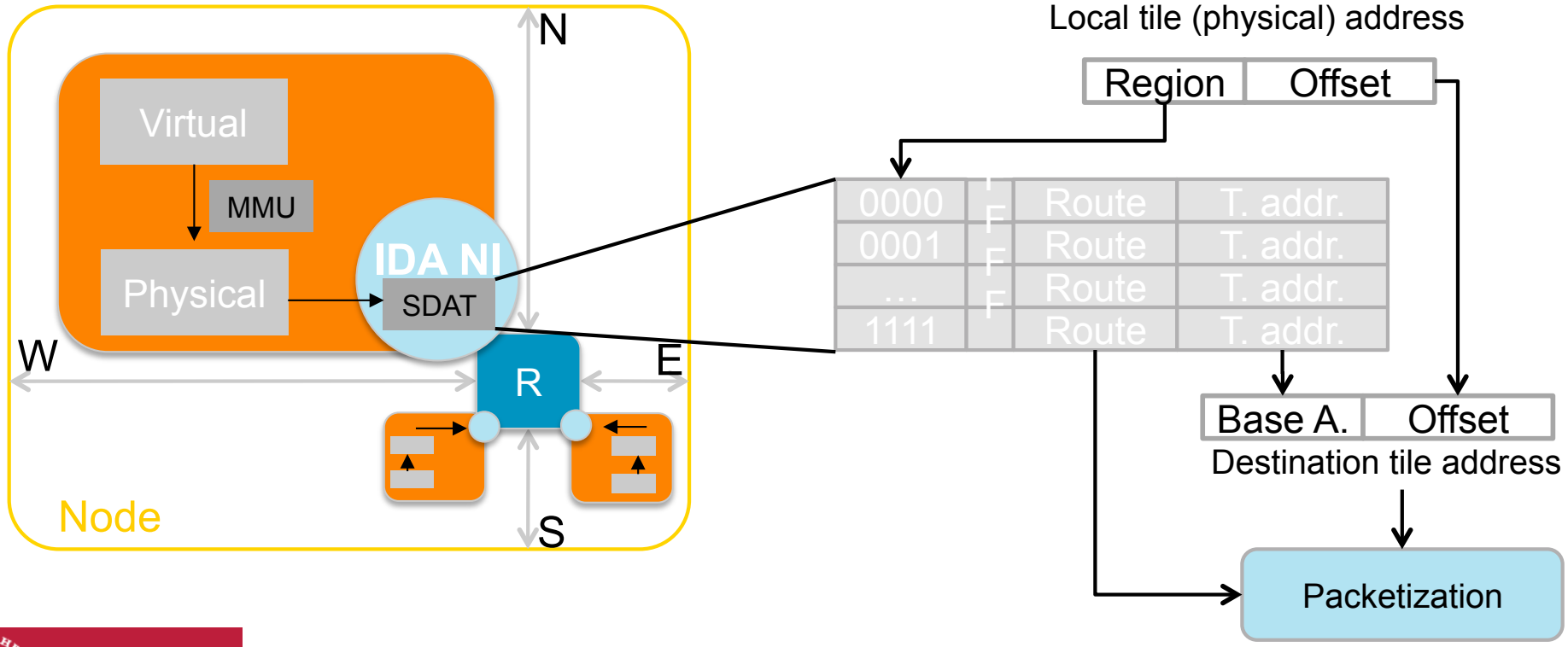
Network Interface (NI) - Architecture



- interface to AMBA bus
- system-virtualization of remote resources
  - Address remapping
  - Interrupt mapping
  - Routing
  - Virtual Channel selection
- configured by trusted central system controller
- monitoring
  - error detection and isolation

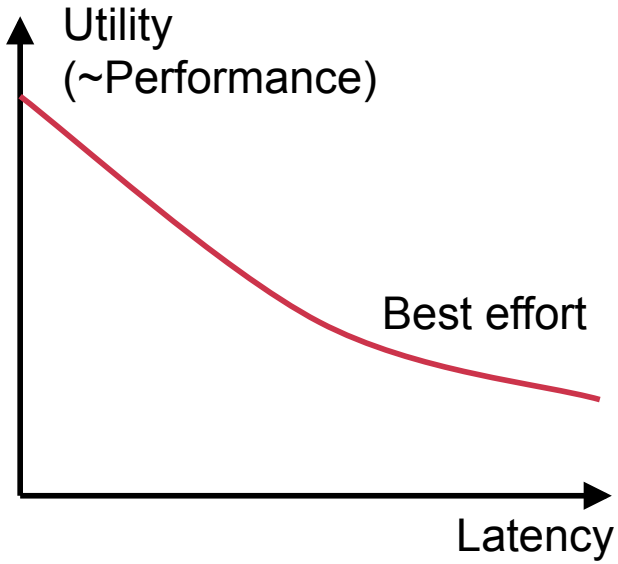
System domain address translation

- translate tile-local physical address to system-wide address
  - flags to limit access (e.g. read-only)
  - route = address to remote tile
  - target address = base address in remote tile

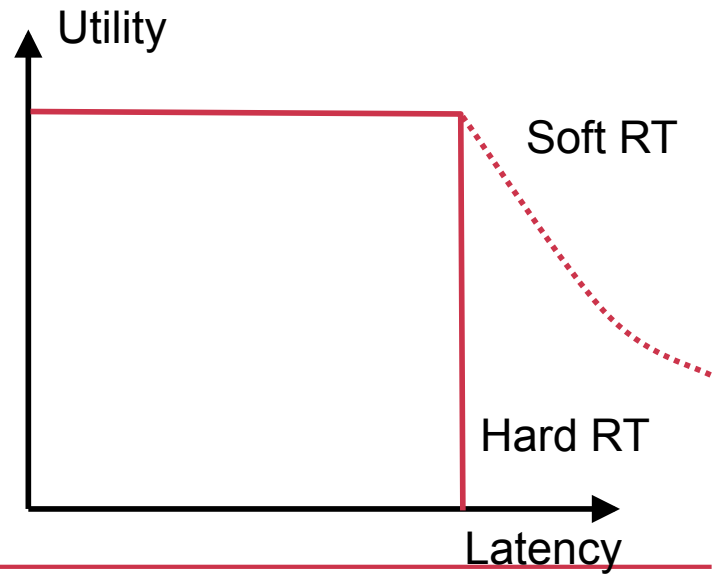


IDAMC – NoC
characteristics of application/traffic classes

- **best-effort applications**
  - **most existing applications, major role in user experience**
  - **unpredictable and bursty resource usage**
  - **latency-sensitive: Application performance degrades with higher latency**

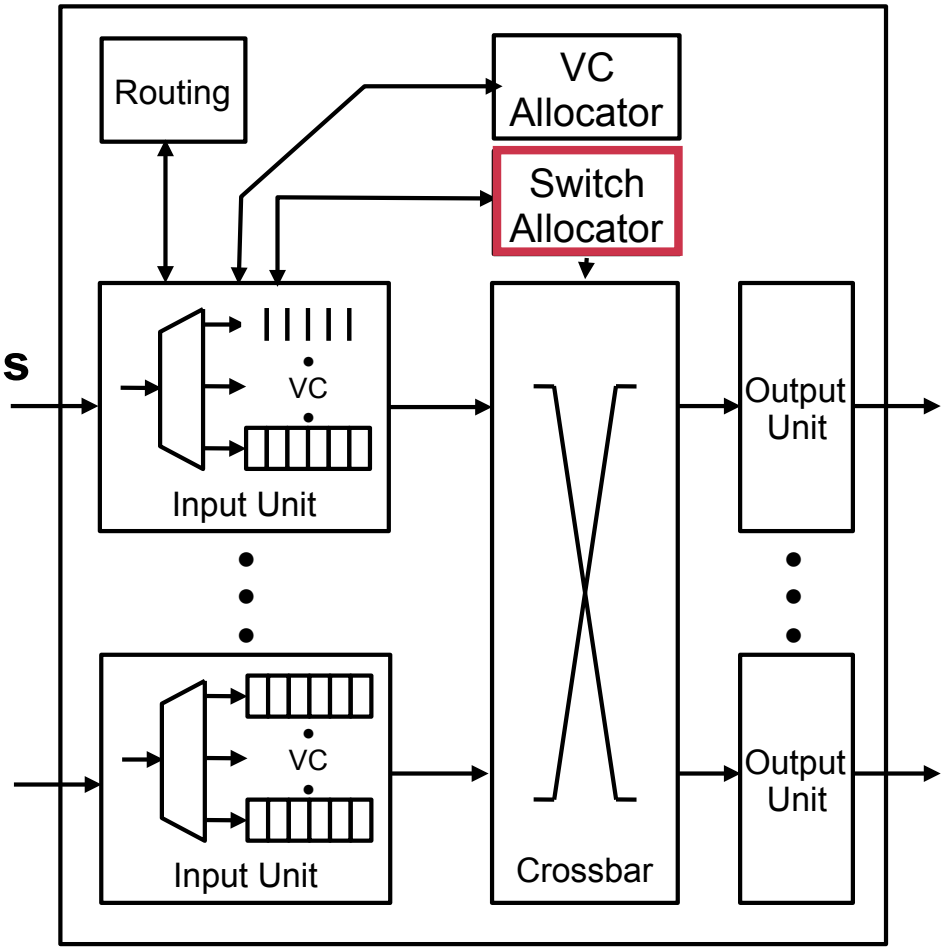


- **real-time streaming applications**
  - **require resource and timing guarantees**
    - **resource sharing must be under control for efficient co-execution**
  - **regular access patterns → latency-tolerant: performance does not degrade with higher latency (up to a certain latency bound)**



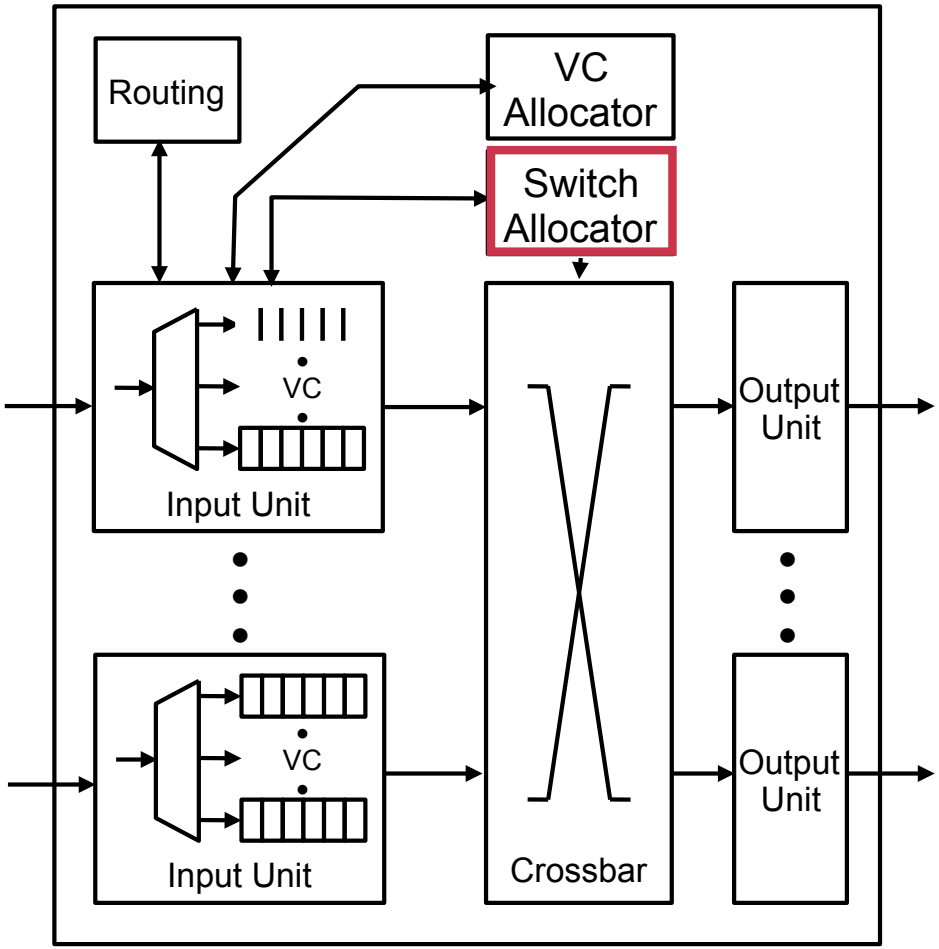
QoS-Support for traffic isolation in the router 1/2

- manage contention at router's outputs
  - static isolation
    - e.g. time-division multiple-access
      (AEthereal [Goossens], SuperGT [Marescaux])
    - service independent of other streams
  - dynamic isolation
    - e.g. prioritization
      (MANGO [Bjerregaard], QNoC [Bolotin], [AlFaruque], Globally-Synchronized Frames [Lee])
    - service depends on the behavior of other streams



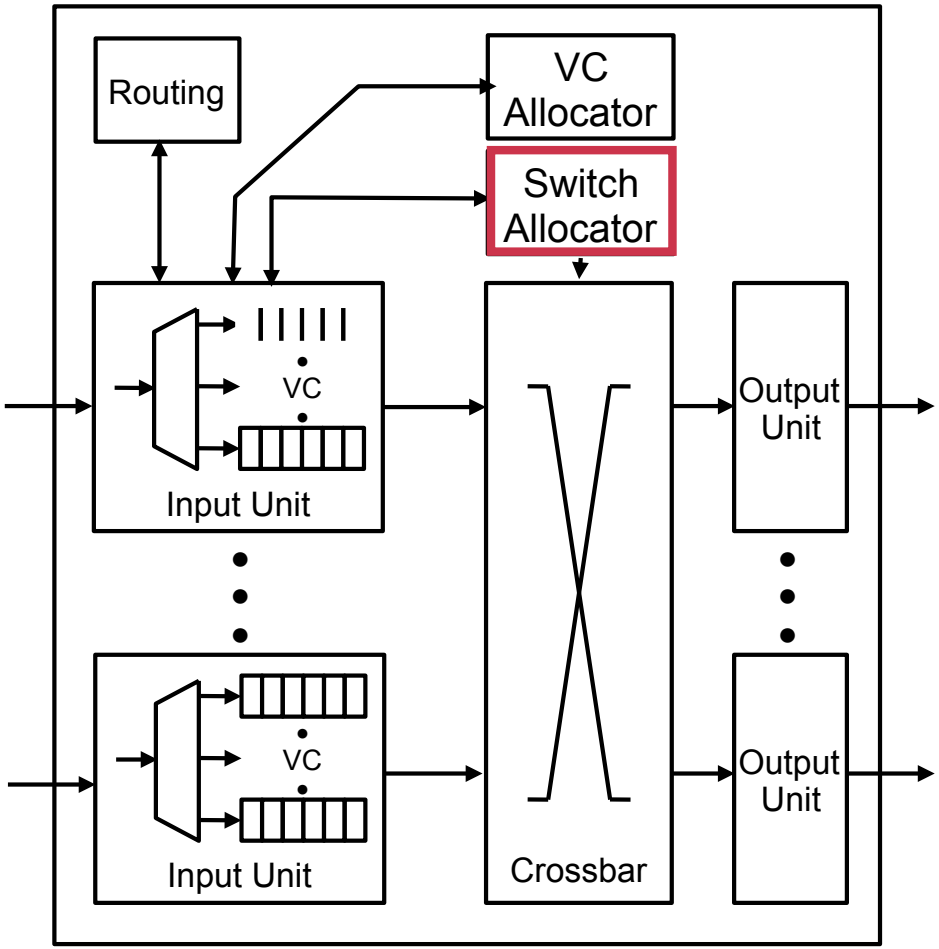
QoS-support for traffic isolation in the router 2/2

- existing QoS do guarantees first!
  - best-effort traffic = “second-class citizen”
    - BE traffic suffers from high latency
    - RT traffic has no benefit from reduced latency (deadline driven)

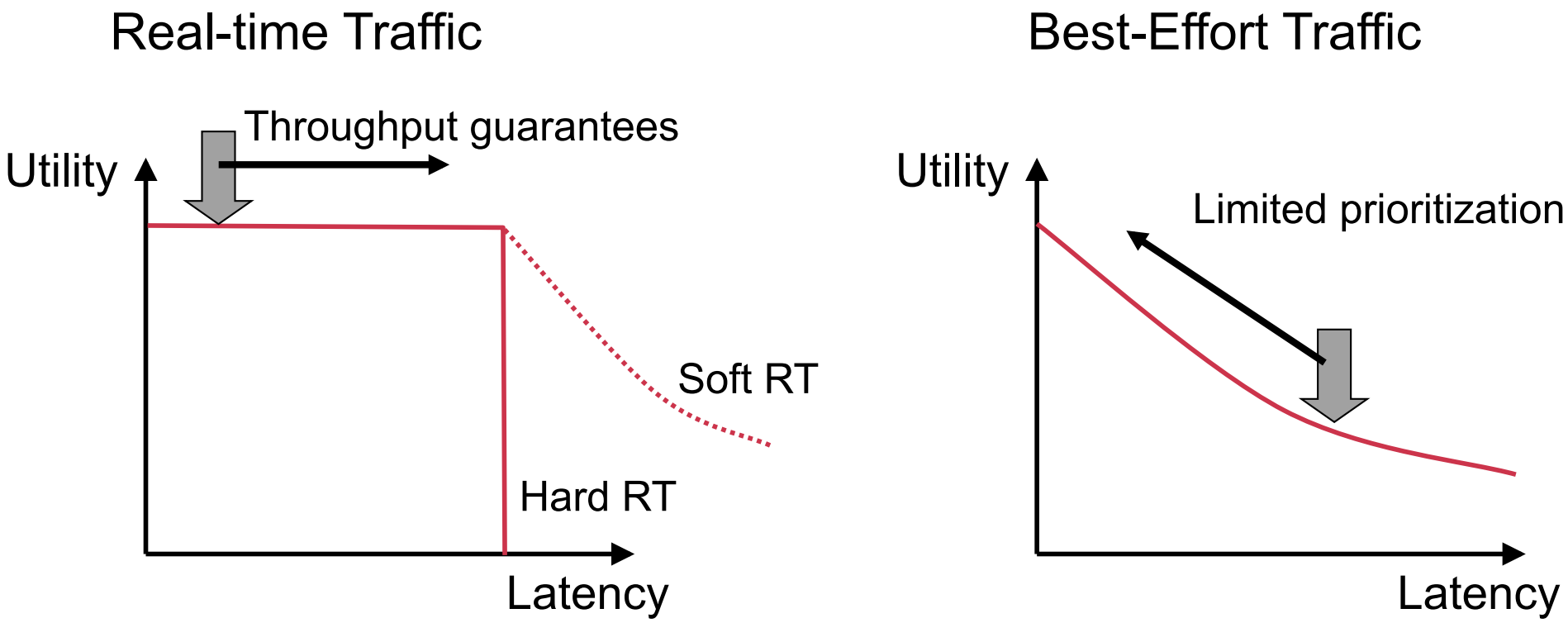


Solution: QoS support for latency sensitive traffic

- idea:
  exploit latency tolerance of RT streaming applications to improve BE latency
- approach: prioritize BE as long as guaranteed throughput (GT) traffic makes sufficient progress
  - Distributed Traffic Shaping (DTS)
  - Back Suction (BS)

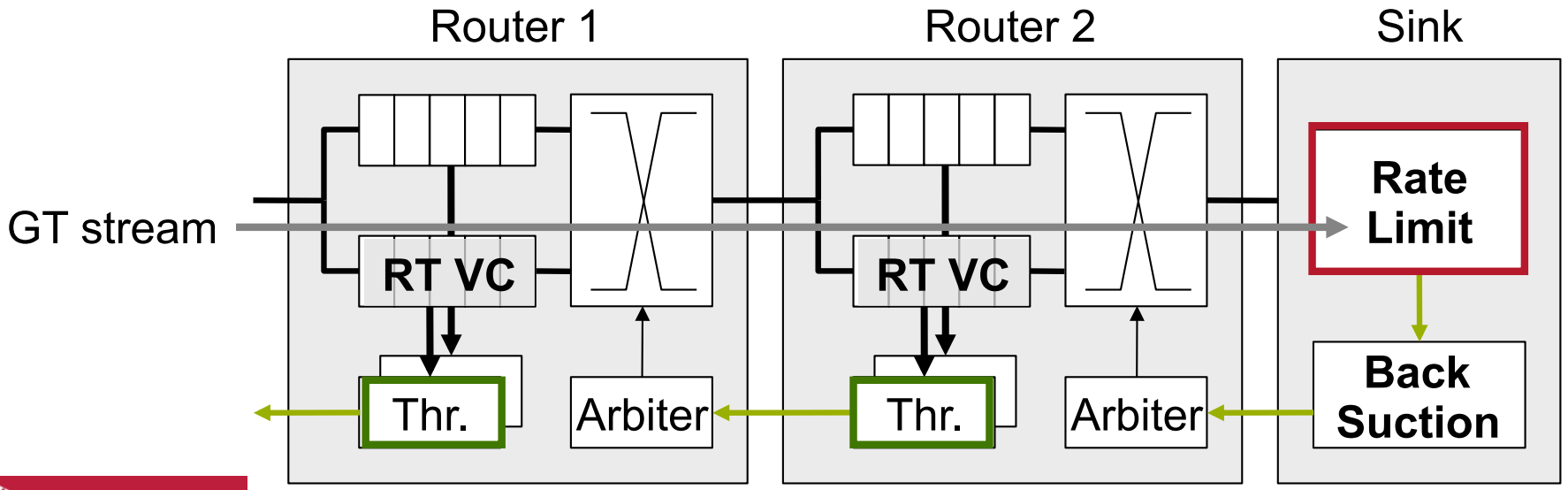


Goal: Guarantees and low BE latency



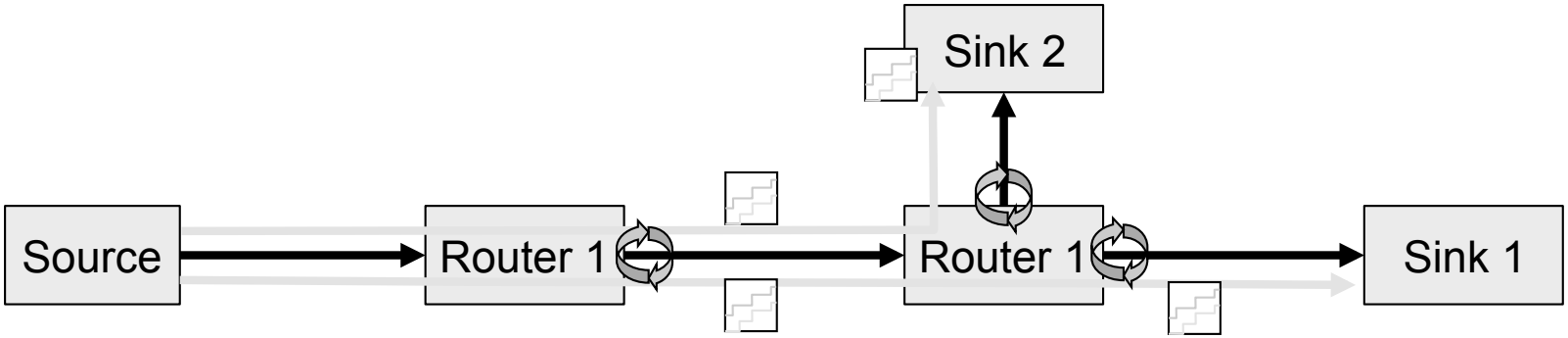
Back Suction (BS)

- Prioritize RT traffic based on downstream buffer occupancy
- Threshold Module at every VC
  - Forward back suction signal on low occupancy towards upstream
  - Threshold determines how early prioritization of RT propagates towards sink
- Limit rate (to guaranteed rate) at which sink may assert back suction

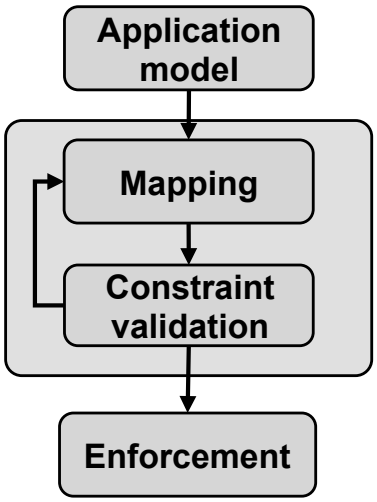


Formal timing analysis for BS and DTS

- formally analyze routers iteratively (starting at sink)

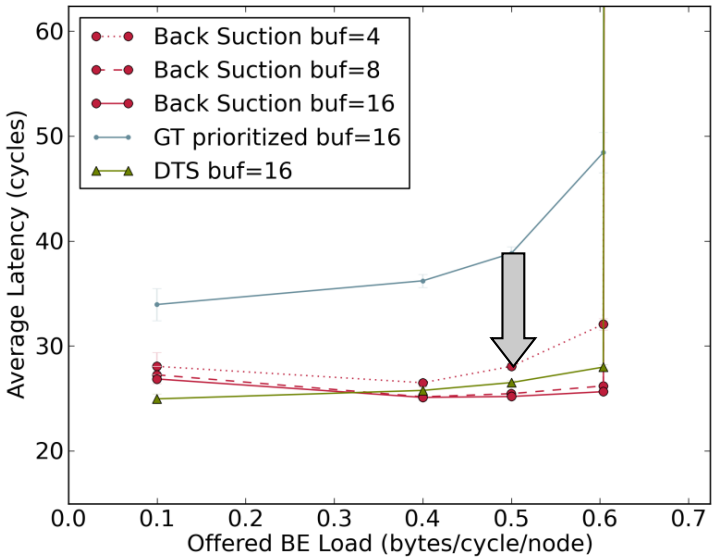


- analysis guarantees GT timing if back suction enforced
  - uses Compositional Performance Analysis
    - based on SymTA/S tool
- future work: admission control performed on-line as part of resource management process



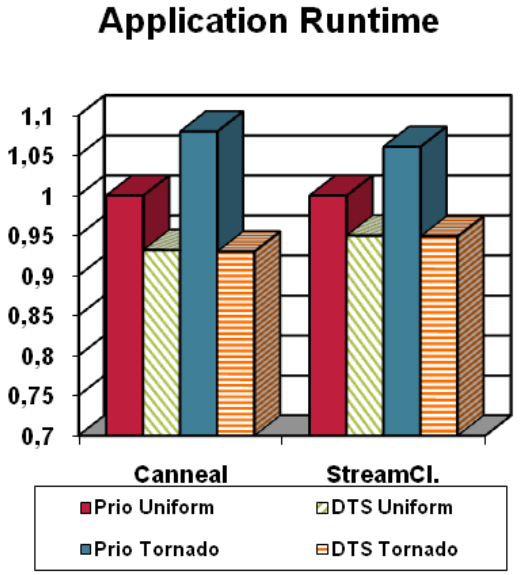
Result: Guarantees and improved BE latency

- mechanism provides throughput guarantees to individual real-time streams
- BE latency is improved significantly
  - application runtime improves accordingly



~ 30% latency improvement over standard prioritization scheme

Improve application performance by >10%



IDAMC - Summary

- efficient separation of critical NoC traffic with minimized negative effect on non-critical traffic
  - back suction gives priority to non-critical traffic granting the requirements of critical traffic
  - exploits full qualification of critical process that includes accurate requirement definition
- include resource access protection
  - external access control protects against accesses from non-critical applications
  - overutilization by non-critical applications constrained by NoC QoS control
- consequence
  - all NoC components and central access and traffic control must be qualified at the highest criticality level in the system

Overview

- motivation
- safety critical embedded system design
- networks and multi-core systems for mixed time and safety critical applications
- **reliable systems for higher safety requirements**
- example projects
- conclusion

Technology trends – Reliability issues

- reliability is an important challenge in future technology generations
  - growing system complexity combined with continuous technology downscaling → increasing error rates
- appropriate techniques necessary to prevent failures
  - *fault isolation*
  - *error detection and correction*
    - *bus/network:* message retransmission, forward error correction
    - *CPU/ECU:* redundancy, rollback techniques, microarchitectural measures
- *problem:* predictability of system reliability
  - how does the system behave in case of errors?
  - what are consequences for the user / for the environment?
  - what is the failure probability?

Fault handling for higher criticalities

- for higher safety requirements separation is necessary but not sufficient
  - isolation still requires correct hardware function
  - hardware failures must be included when hardware is less reliable than safety requirements
  - embedded systems trend
    - reliability of technology ↓
    - safety requirements ↑
    - *hot industrial topic!*

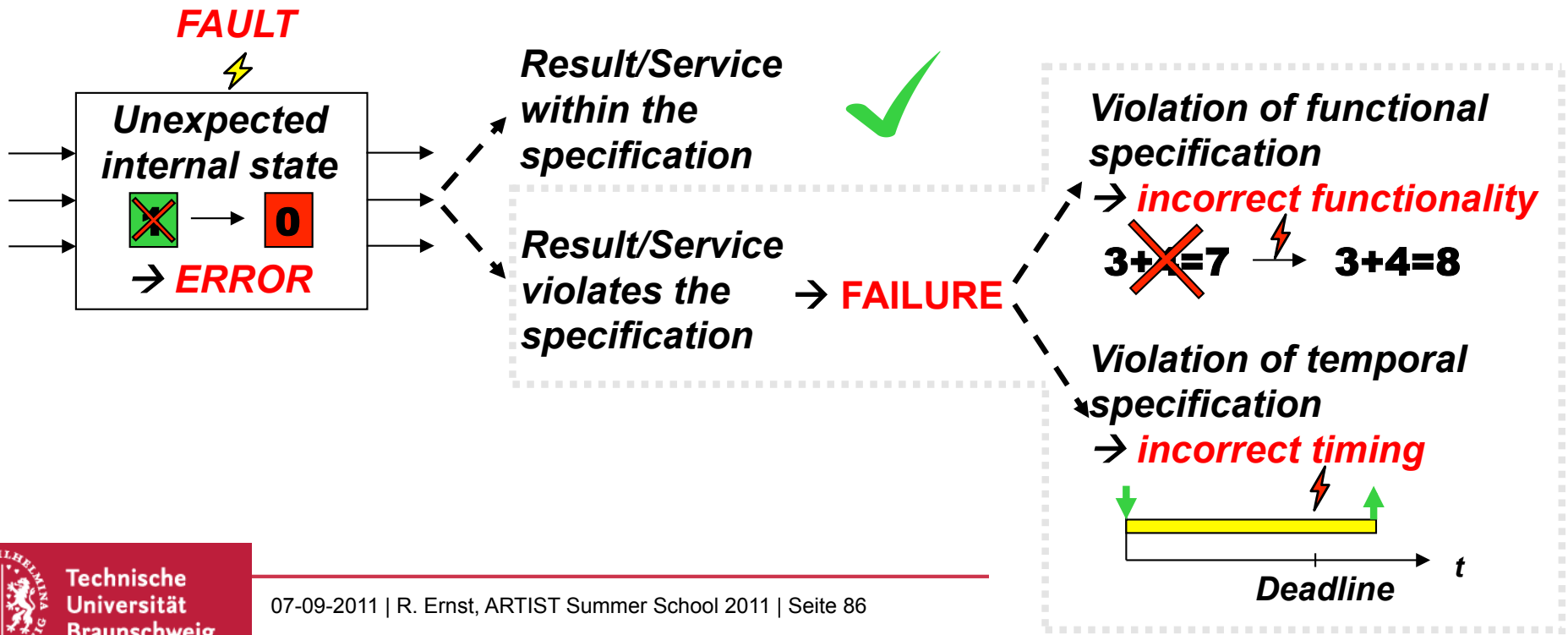
Fault handling for higher criticalities

- handling of static and transient hardware faults required
- reliability requirements are often quantified
  - requires *predictable failure bounds*
- IEC 61508

SIL	*Low demand mode: average probability of failure on demand*	*High demand or continuous mode: probability of dangerous failures per hour*
1	$> 10^{-2}$ to $< 10^{-1}$	$> 10^{-6}$ to $< 10^{-5}$
2	$> 10^{-3}$ to $< 10^{-2}$	$> 10^{-7}$ to $< 10^{-6}$
3	$> 10^{-4}$ to $< 10^{-3}$	$> 10^{-8}$ to $< 10^{-7}$
4	$> 10^{-5}$ to $< 10^{-4}$	$> 10^{-9}$ to $< 10^{-8}$

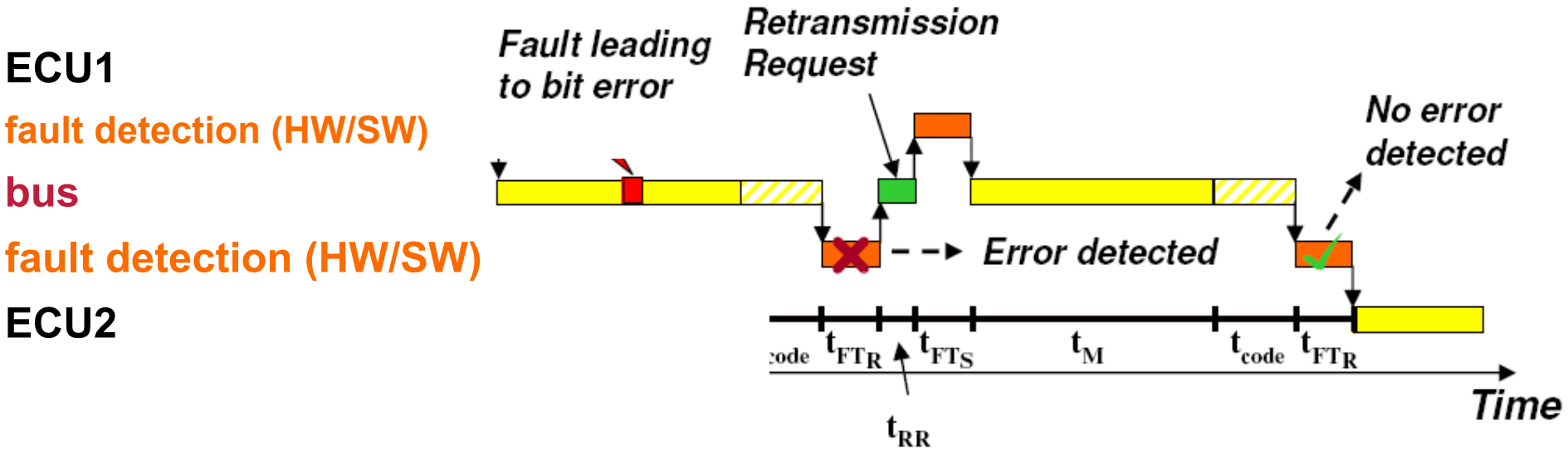
From ES faults to ES failures

- distinguish static and transient ES errors
  - static errors have permanent effects requiring redundancy for repair
  - transient errors are usually more frequent (EMC, new semiconductor technologies, …) but can often be masked when detected
    - transient error masking can cause timing errors

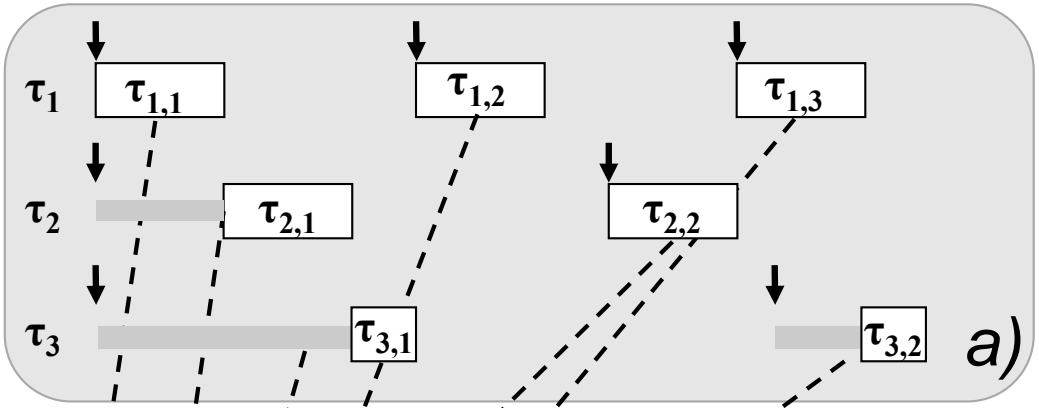


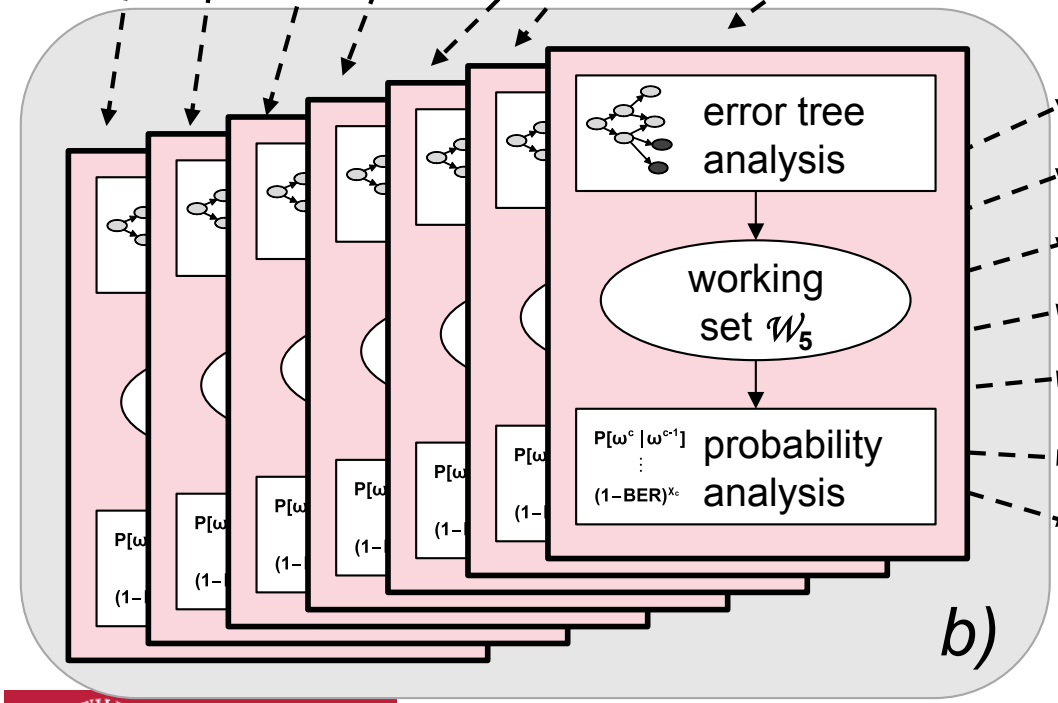
Transient error handling

- transient error handling known from communication
  - example: CAN (automotive)
- CAN has error detection capabilities (CRC)
  - repeats message in case of transmission error using defined protocol
  - *CAN functional fault tolerance increases timing and load!*



Reliability analysis – General concept

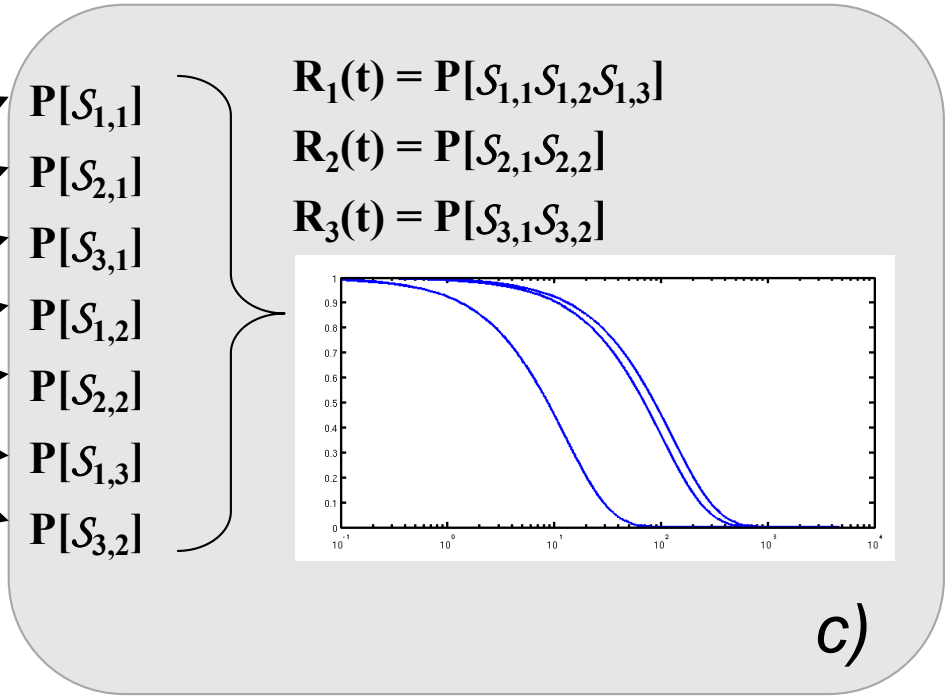




a) timing prediction

b) success probability calculation

c) reliability probability composition



CAN - Example presented at last Summer School

- SAE benchmark frame set
  - periodic system, deadline end of period
  - approximately 5 x 10^6 activations per hour
  - bus load approx. 70 % (CAN at 150 kbit/s)
- error model
  - BER = 10^{-7} [Ferreira, 2004]
  - residual errors according to [Charzinksi, 1994]

$MTTF_{func}$	2×10^{12} h	SIL 4
$MTTF_{time}$	$1{,}8 \times 10^{5}$ h	SIL 1

7 orders of magnitude more likely to miss end-of-period deadline!
(for single error fault model)

Lessons to be learned from communication

- functional errors can efficiently be detected
  - redundancy in time (message resend) is efficient
- timing failures can be much more likely than residual errors
  - even at moderate load
  - due to failures at points with peak load
- *consider peak load situations!*

Reliability in multi-core systems

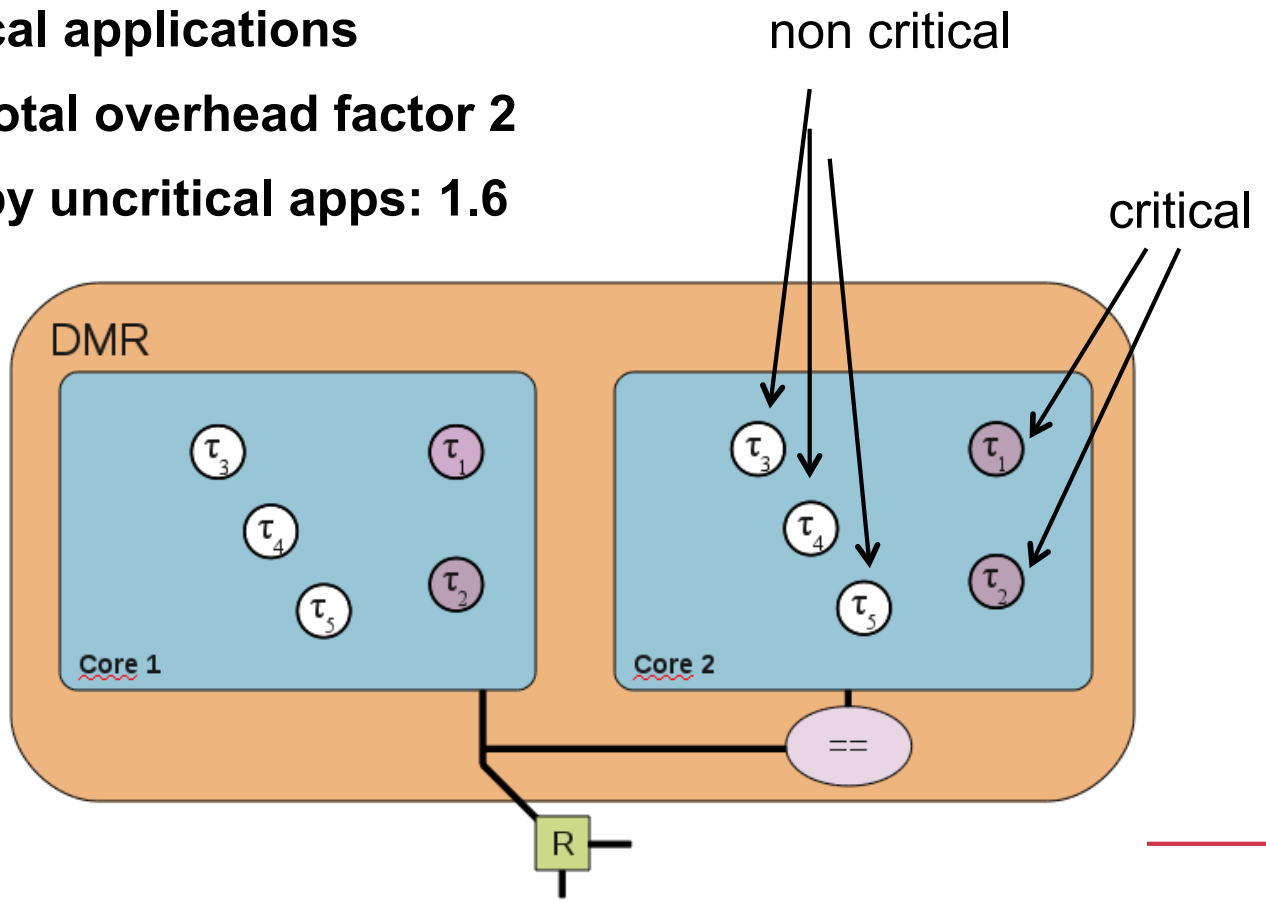
- communication can easily incorporate fault-tolerance
  - EDC + retransmission (CRC, parity, hashes)
  - ECC (Hamming, Turbo, …)
- computation is much harder to protect
  - entire processor affected
  - control and data flow (including IP cores)
    → errors can propagate

VLSI failure modes and fault-tolerance

- different failure modes of transistors
  - permanent errors (electromigration → stuck-at-error)
  - soft errors (SEUs, cosmic-radiation, thermal neutrons)
  - stress induced transient errors (transistor variability, NBTI, PBTI → V_t shift)
- soft errors and transient effects are expected to dominate
- all fault tolerance methods are based on redundancy
  - spatial redundancy
    - dual modular redundancy (DMR): (single) errors will be detected
    - triple modular redundnacy (TMR): (single) errors can be corrected
  - temporal redundancy (e.g. reevaluation on the same processor)
    - slightly worse coverage (permanent errors not detected)
    - high impact on timing (additional workload)
    - can also be used for recovery (checkpointing & rollback)

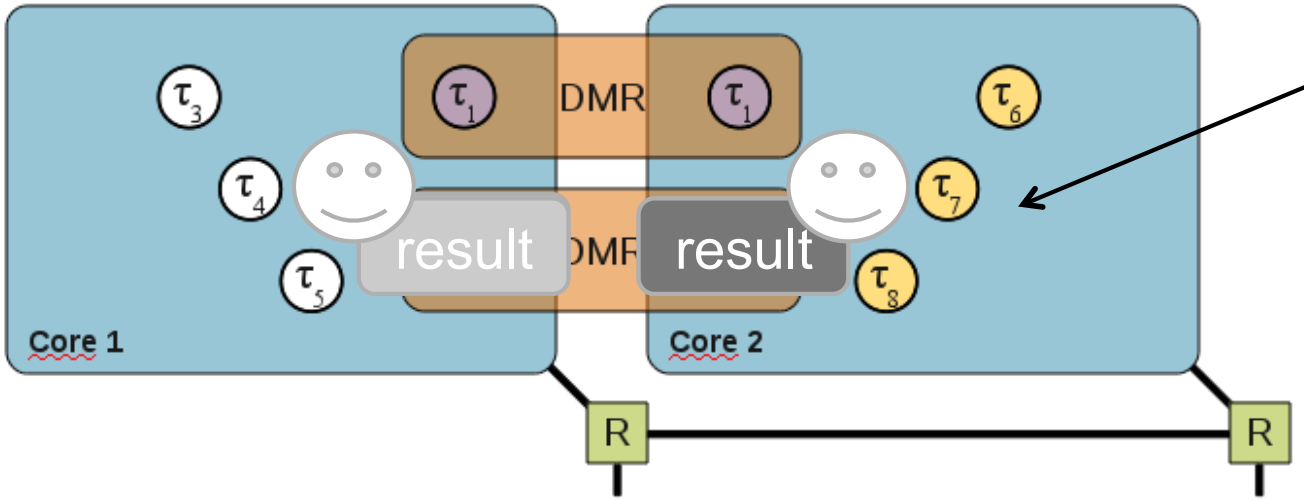
Traditional DMR (e.g. lockstep architecture) 2/2

- DMR does not scale for mixed-critical systems
  - not all applications need fault tolerance
- mixed-criticality example
  - 2 critical, 3 uncritical applications
  - system lockstep: total overhead factor 2
  - overhead caused by uncritical apps: 1.6



Task level DMR (fine grained)

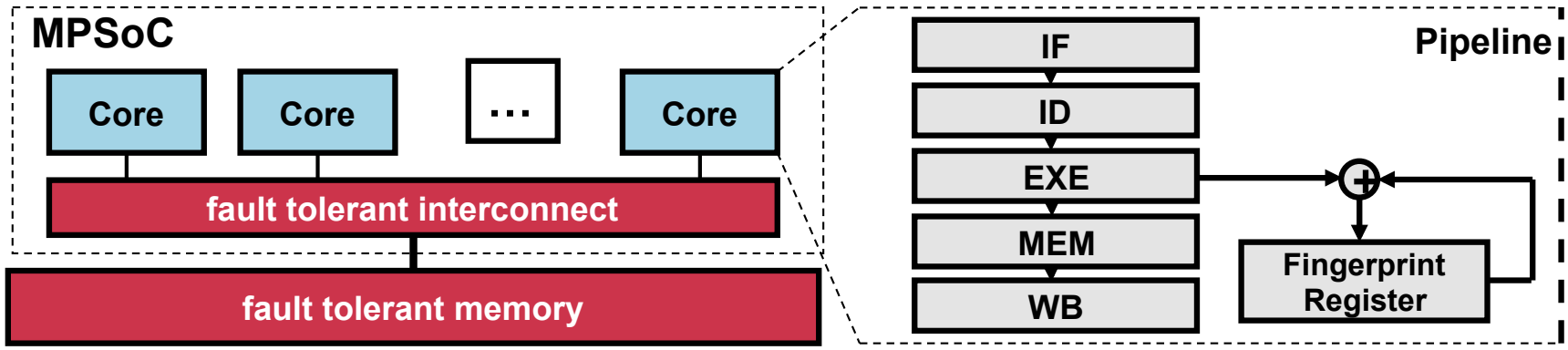
- replicate individual tasks only where needed
- comparisons is based task state (e.g. comparison at output)
- result can be verified if results from both instances are available
  - DMR creates feedback from core1 on core2 and vice versa
- redundant copies induce higher load on other tasks
- additional overhead through core-to-core communication (comparison)
- timing analysis is necessary to prove correctness



free resources can be used by other uncritical apps

Fingerprinting

- compact state to “fingerprint” reduce core-to-core communication and comparison overhead
- compare **execution fingerprints** (Smolens et al., 2004)
  - efficient, low bandwidth, high coverage

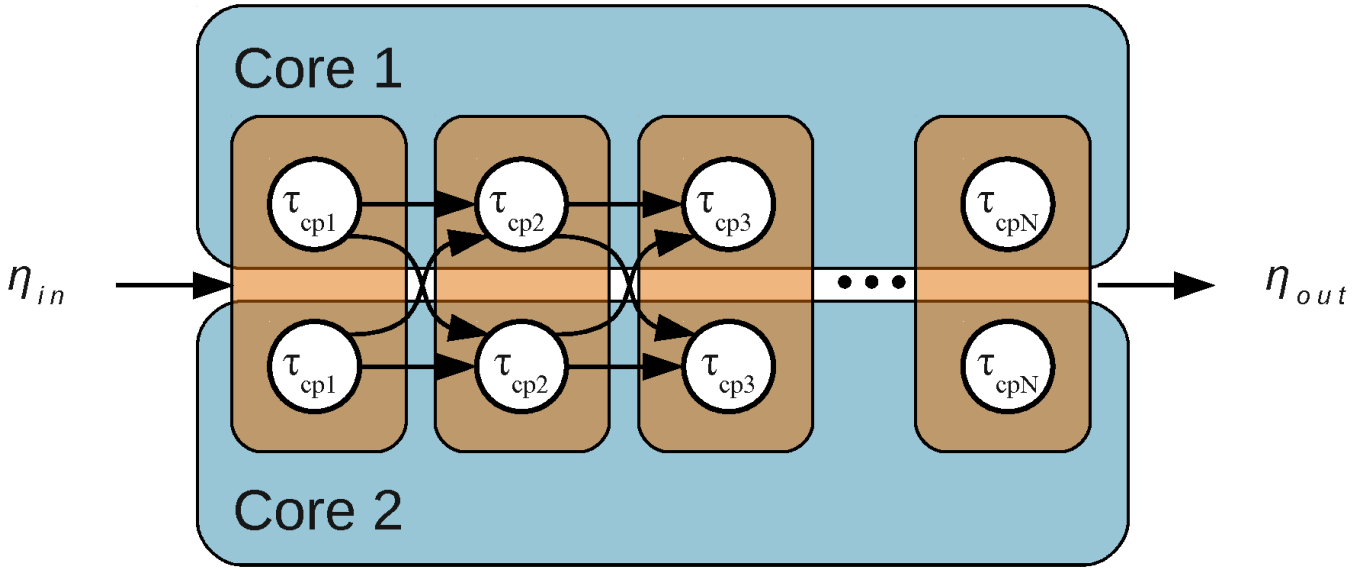


Recovery – Typical methods

- system reset
  - usually infeasible in hard real-time systems
- switch to fail-safe mode (e.g failed active steering → fallback to mechanical solution)
- checkpointing and rollback recovery
  - save state in regular intervals and store it on reliable memory
  - in case of errors, restore most recent state
  - impact on timing on all (lower priority) tasks in case of errors
- recovery methods lead to timing overhead!
  - cp. resend in communication
  - *use timing analysis*

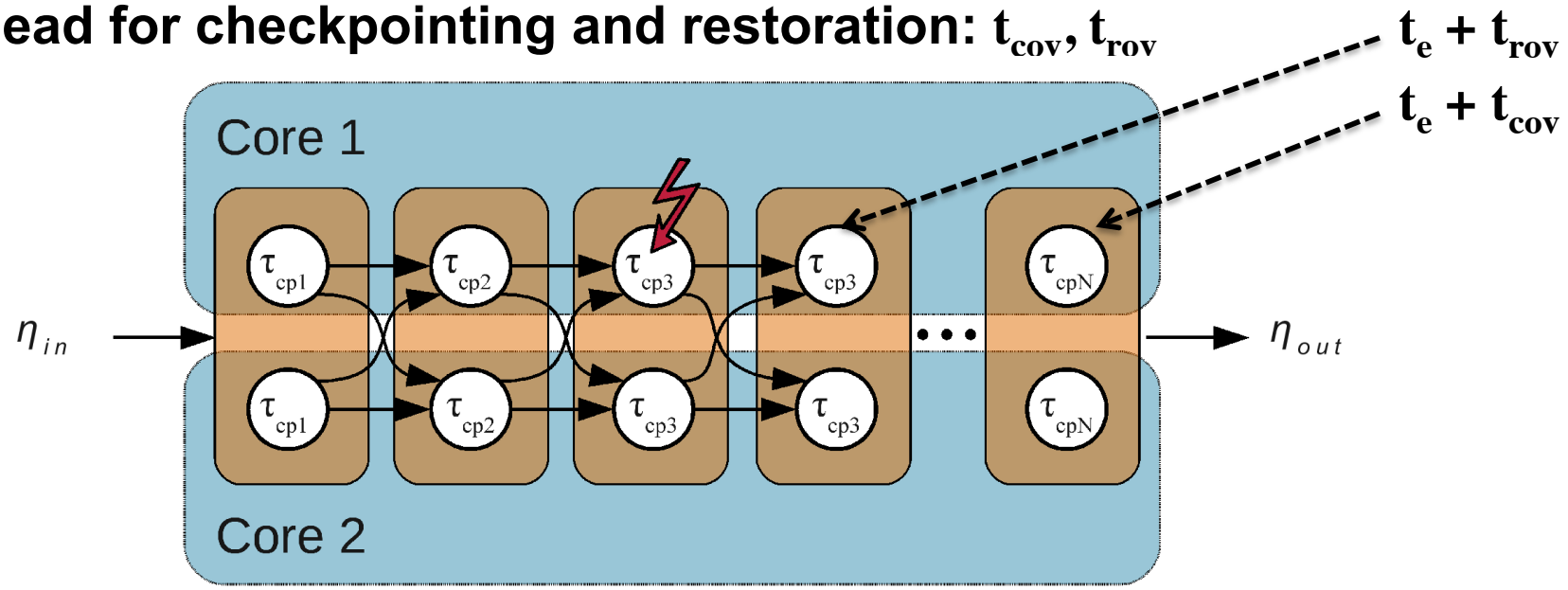
Modeling of fine grained redundancy

- task τ_i is split into n segments
- identical replications of task is distributed among e.g. k cores (e.g. 2)
- voting on intermediate result (fingerprint) is performed in arbitrary (but known) intervals
- use an equivalent task graph as a model

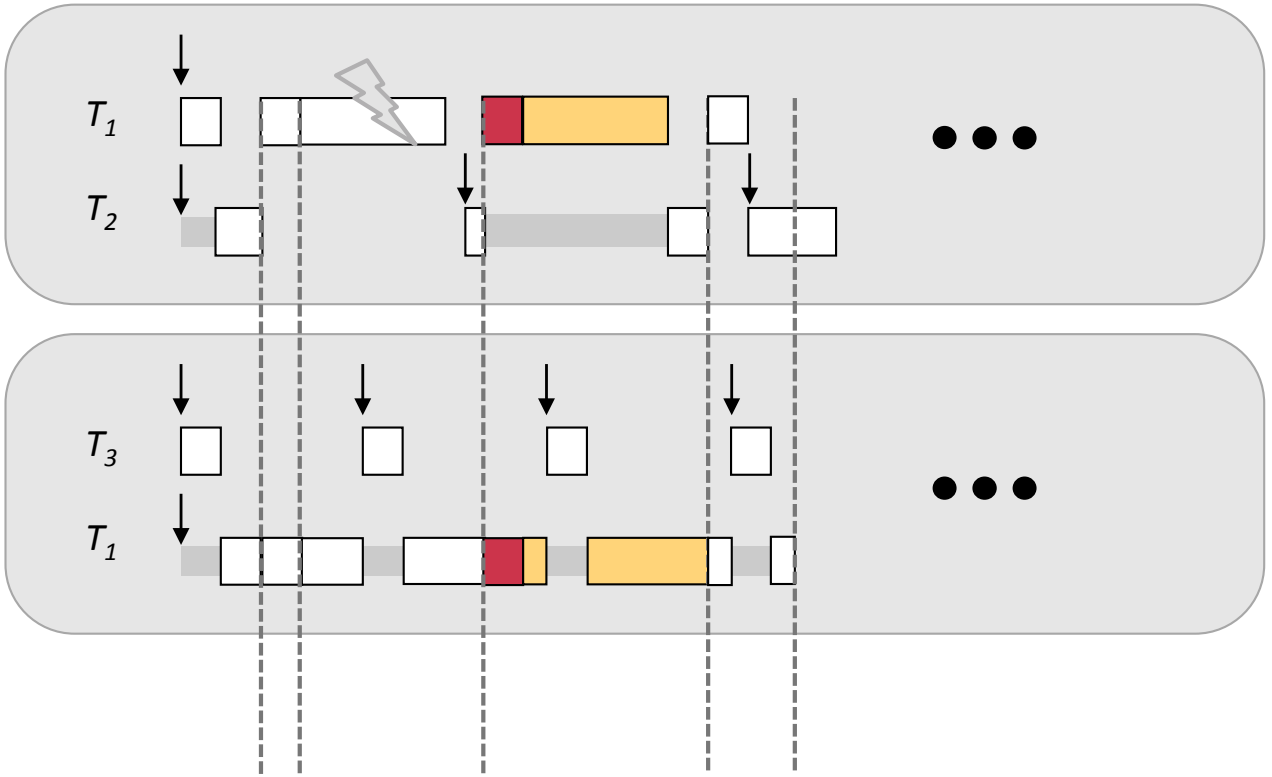


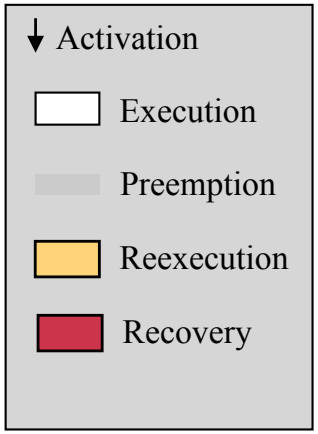
Recovery operations

- non-matching fingerprints → recovery required
  - we model checkpointing and rollback
  - right before each segment, the state is saved to ECC protected memory
  - in case of an error, a correct state is restored
  - overhead for checkpointing and restoration: t_{cov}, t_{rov}



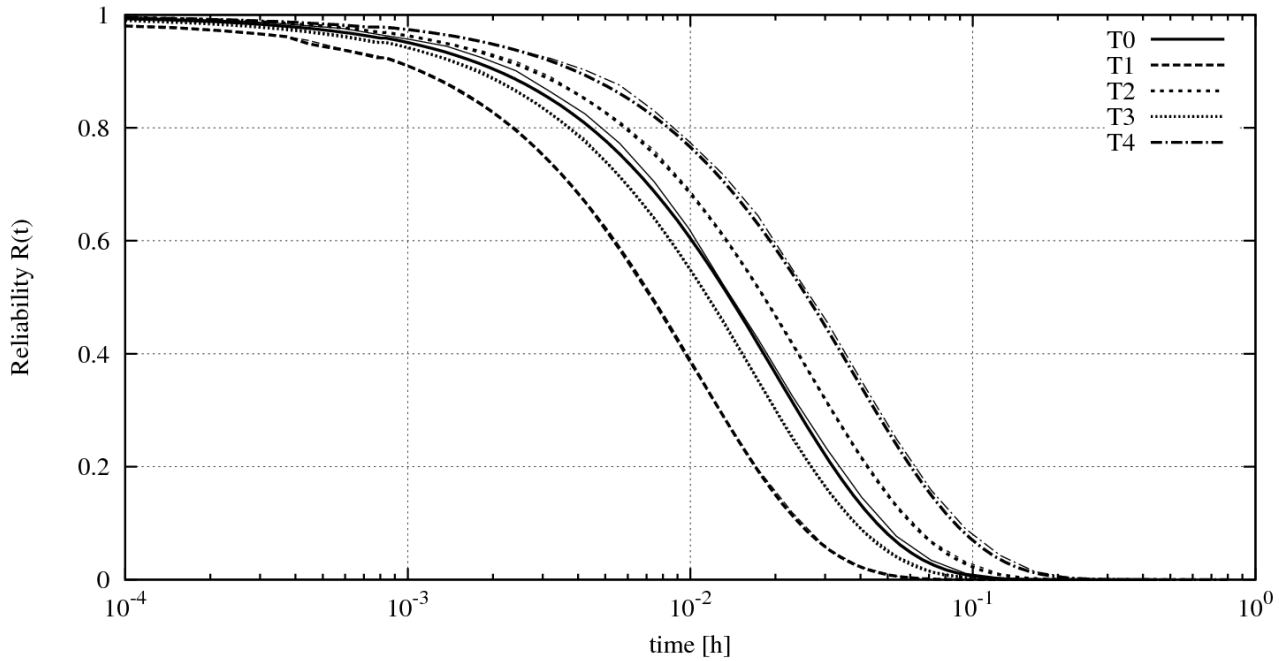
Timing failures – Re-execution example (simplified)





comparison of intermediate results

Accuracy – comparision with Monte Carlo analysis



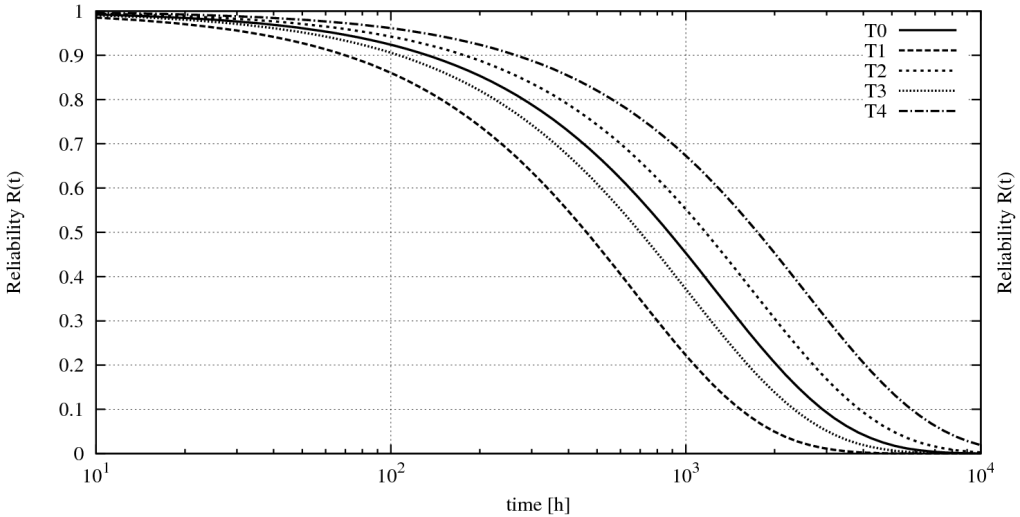
$\lambda_{1,2} = 1/10$ sec

Overhead
- Checkpoint Creation
- Recovery

Task	Priority – Mapping core1/core2	T	C	Checkpoints	Creation	Recovery
T0	3/2	300	60	2	10	20
T1	4/-	250	50	-		
T2	2/-	100	10	-		
T3	-/1	300	50	-		
T4	1/3	600	40	2	10	20

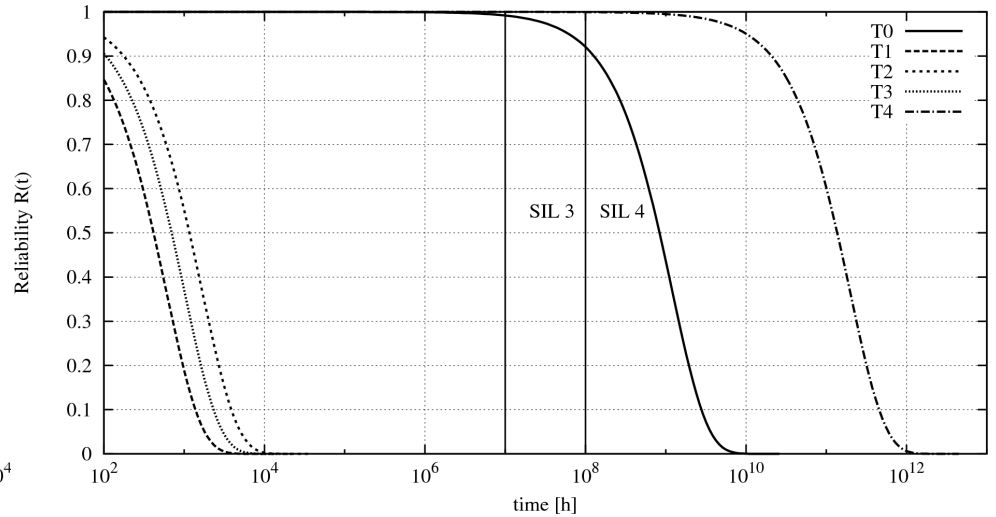
Design tradeoffs - Checkpointing

λ = 1/week



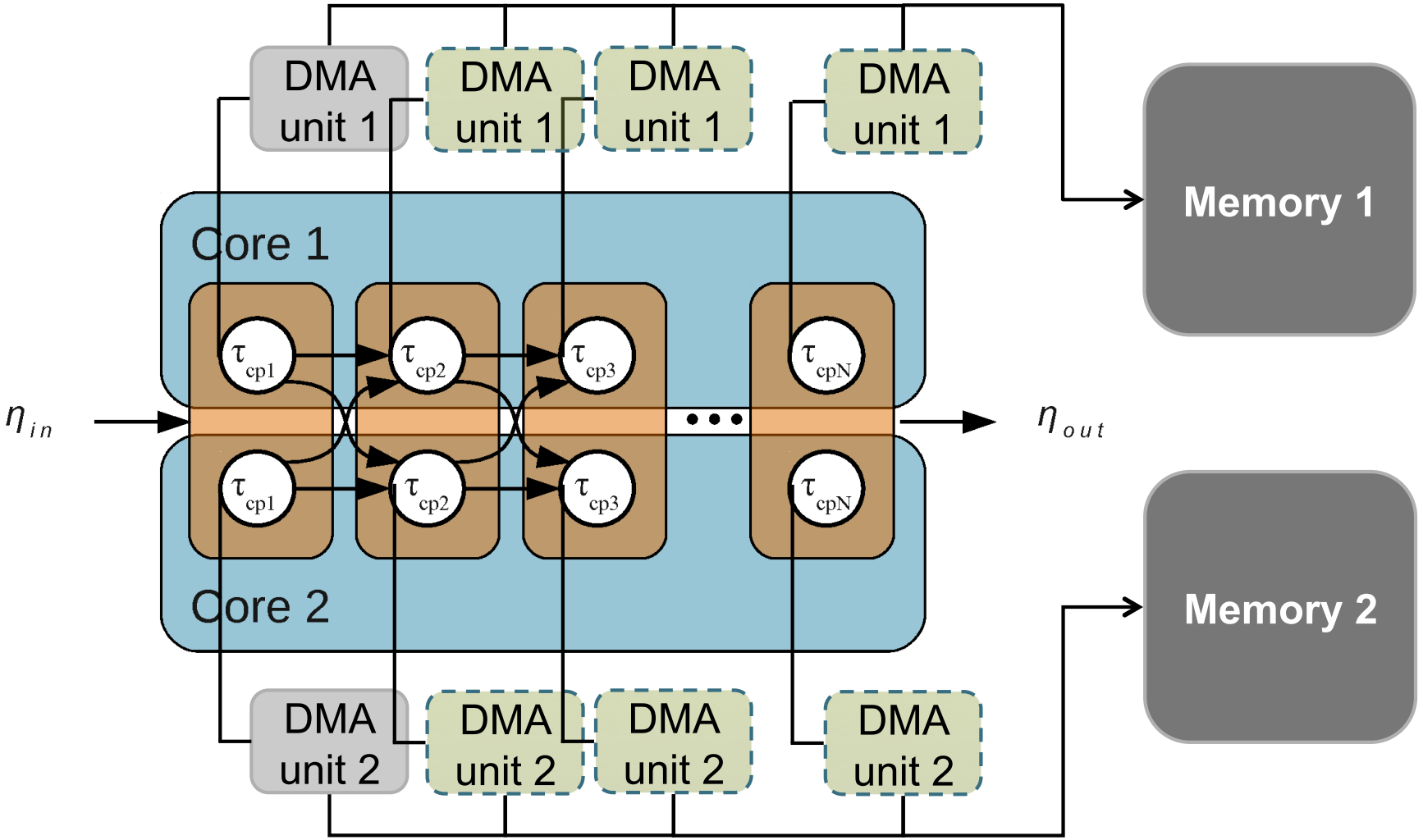
- checkpointing itself can be a „critical section“
- for this experiment we assume, that an error during checkpointing causes a system failure

λ = 1/week



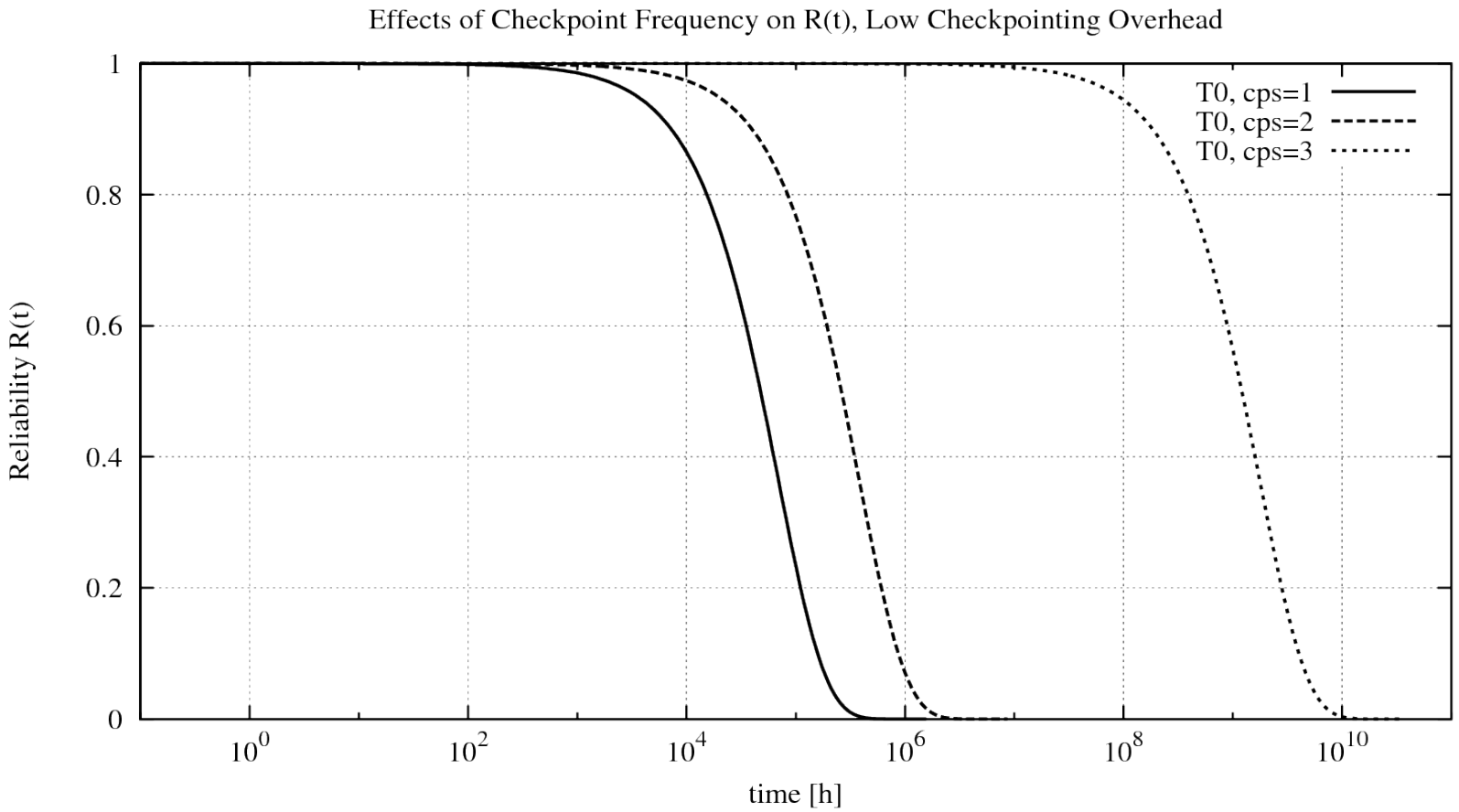
- assume fault-tolerant checkpointing
- implementation
  - e.g. ECC memory protection & bus EDC
  - dual (redundant) DMA engines

Redundant DMA engines for checkpointing



checkpoint is copied by two different DMA engines to different memories

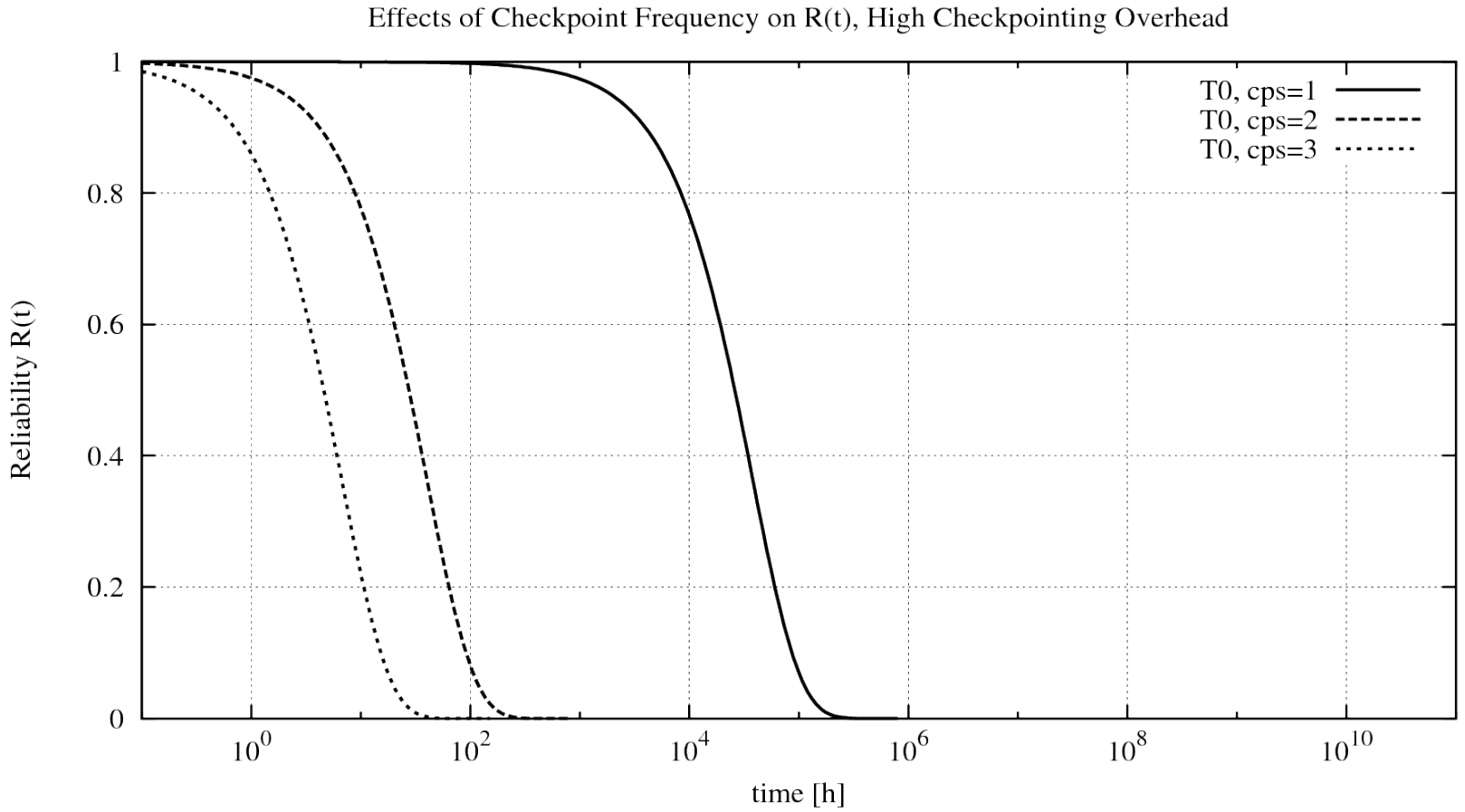
Tradeoffs – No. of checkpoints 1/2



- **experiment 1: low checkpointing overhead: 0.1ms**

  → R(t) increases with the amount of checkpoints

Tradeoffs – No. of checkpoints 2/2



- **experiment 2: high checkpointing overhead: 1ms**

  → R(t) decreases with the amount of checkpoints

The cost of safety

- multi-core for highly safety critical functions require major hardware overhead
- systems with mixed criticality bare the risk of non manageable requirements
  - vastly different design quality for safety and non-safety function integrated on the same platform
  - function isolation must be complemented by costly fault tolerance
- efficient methods for function isolation and predictable fault tolerance are needed
- example: Project RECOMP

Overview

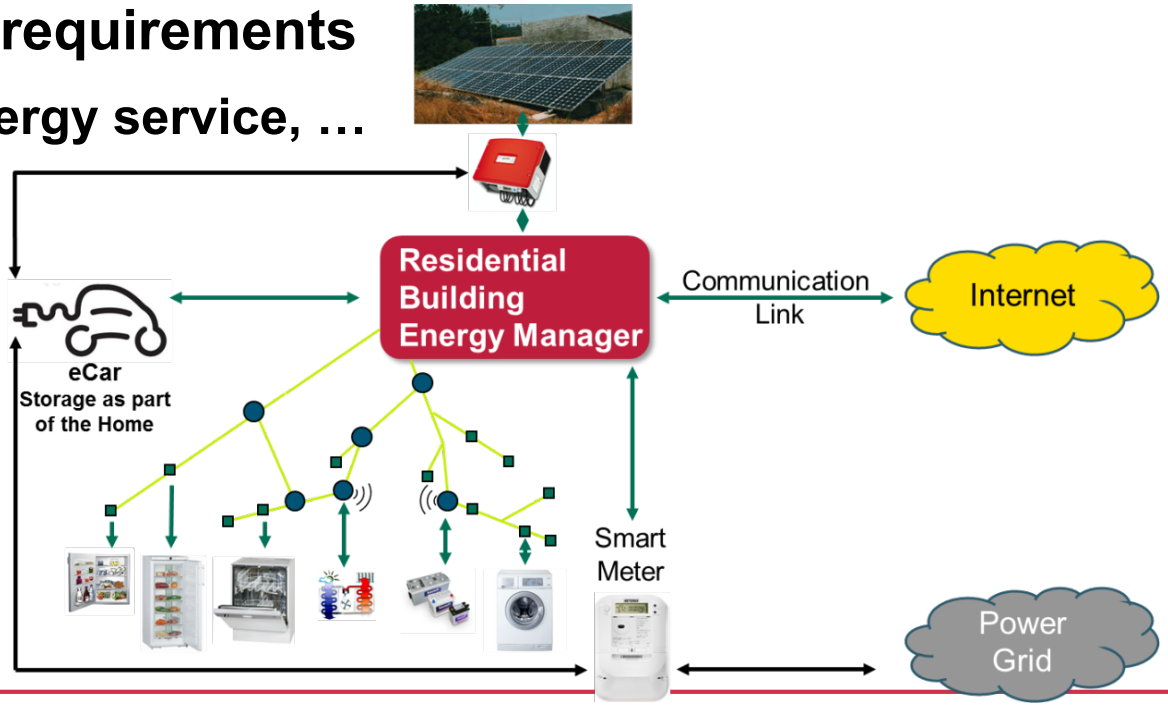
- motivation
- safety critical embedded system design
- networks and multi-core systems for mixed time and safety critical applications
- reliable systems for higher safety requirements
- **example projects**
- conclusion

RECOMP

- "Reduced Certification Costs Using Trusted Multi-core Platforms"
- multi-core architectures for mixed safety critical systems
  - flexible tradoff between development, certification, and hardware cost
  - requires configurable core-to-core communication and separation/virtualization technologies
- objective
  - develop HW and SW architectures, design methods, and tools to efficiently design and (re-)certify MpSoCs for mixed critical systems
- European ARTEMIS project, 41 partners, 25 Mio € budget, 2010-2013
  - covers whole design chain
    - semiconductors, RTOS, suppliers, integrators (OEMs)
  - several industries
    - automotive, aerospace, industrial
- *www.recomp-project.eu*

IoE

- "Internet of Energy (IoE) for Electric Mobility"
- objective
  - develop hardware, software and middleware to use the Internet for future smart energy grids incl. new applications (electric mobility)
- European ARTEMIS project, 42 partners, 45 Mio € budget, 2011-2014
- complex mixed criticality requirements
  - grid stability, individual energy service, …
  - economic security
- large economic challenges
  - power efficiency
  - network efficiency …



Overview

- motivation
- safety critical embedded system design
- networks and multi-core systems for mixed time and safety critical applications
- reliable systems for higher safety requirements
- example projects
- **conclusion**

Conclusion

- merging safety critical functions on an embedded platform typically leads to mixed critical platform components
- mixed criticality is a serious certification cost driver and limits update capabilities of non-critical functions
- no silver bullets for integration available
- potential integration technologies span broad design space but still lack coherence and completeness
- project consortia dedicated to the mixed criticality challenge in local (RECOMP) to widely distributed (IoE) systems
- much further research needed

Thank you!

Acknowledgements

- the following people have contributed to the slides
  - Philip Axer
  - Jonas Diemer
  - Mircea Negran
  - Simon Schliecker
  - Maurice Sebastian

Literature (selected)

- **RECOMP http://www.recomp-project.eu/**
- **for the challenge of multi-core performance dependencies see**
  - **Mircea Negrean, Simon Schliecker, Rolf Ernst. "Response-Time Analysis of Arbitrarily Activated Tasks in Multiprocessor Systems with Shared Resources." In *Proc. of Design, Automation, and Test in Europe (DATE)*, Nice, France, April 2009.**
- **for BS and DTS**
  - **Jonas Diemer and Rolf Ernst, "Back Suction: Service Guarantees for Latency-Sensitive On-Chip Networks," in *Proceedings of the 4th ACM/IEEE International Symposium on Networks-on-Chip (NOCS'10)*, May 2010**
- **for fault tolerance**
  - **Maurice Sebastian, Philip Axer, Rolf Ernst, Nico Feiertag, und Marek Jersak, "Efficient Reliability and Safety Analysis for Mixed-Criticality Embedded Systems," *SAE System Level Architecture Design Tools and Methods*, April 2011**
  - **Maurice Sebastian, Rolf Ernst, "Reliability Analysis of Single Bus Communication with Real-Time Requirements," in Proc. of 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), (Shanghai, China), November 2009**